23PCSC06 - ADVANCED JAVA PROGRAMMING

UNIT-1: BASICS OF JAVA

Java Basics Review: Components and event handling – Threading concepts – Networking features – Media techniques

UNIT-2: REMOTE METHOD INVOCATION

Remote Method Invocation-Distributed Application Architecture- Creating stubs and skeletons Defining Remote objects- Remote Object Activation-Object Serialization-Java Spaces

UNIT-3: DATABASE

Java in Databases- JDBC principles – database access- Interacting- database search – Creating multimedia databases – Database support in web applications

UNIT-4: SERVLETS

Java Servlets: Java Servlet and CGI programming- A simple java Servlet-Anatomy of a java Servlet-Reading data from a client-Reading http request header-sending data to a client and writing the http response header-working with cookies Java Server Pages: JSP Overview-Installation-JSP tags-Components of a JSP page-Expressions Scriptlets-Directives-Declarations-A complete example

UNIT-5: ADVANCED TECHNIQUES

JAR file format creation – Internationalization – Swing Programming – Advanced java

ADVANCED JAVA PROGRAMMING

UNIT-1

BASICS OF JAVA

Java

Java is a high-level, general-purpose, object-oriented, and secure programming language developed by James Gosling at Sun Microsystems, Inc. in 1991. It is formally known as OAK. In 1995, Sun Microsystem changed the name to Java. In 2009, Sun Microsystem takeover by Oracle Corporation.

Java was developed by James Gosling at Sun Microsystems and it was released in May 1995 as a core component of Sun Microsystems' Java platform.

EDITIONS OF JAVA

Each edition of Java has different capabilities. There are three editions of Java:

- **Java Standard Editions (JSE):** It is used to create programs for a desktop computer.
- **Java Enterprise Edition (JEE):** It is used to create large programs that run on the server and manages heavy traffic and complex transactions.
- **Java Micro Edition (JME):** It is used to develop applications for small devices such as set-top boxes, phone, and appliances.

TYPES OF JAVA APPLICATIONS

There are four types of Java applications that can be created using Java programming:

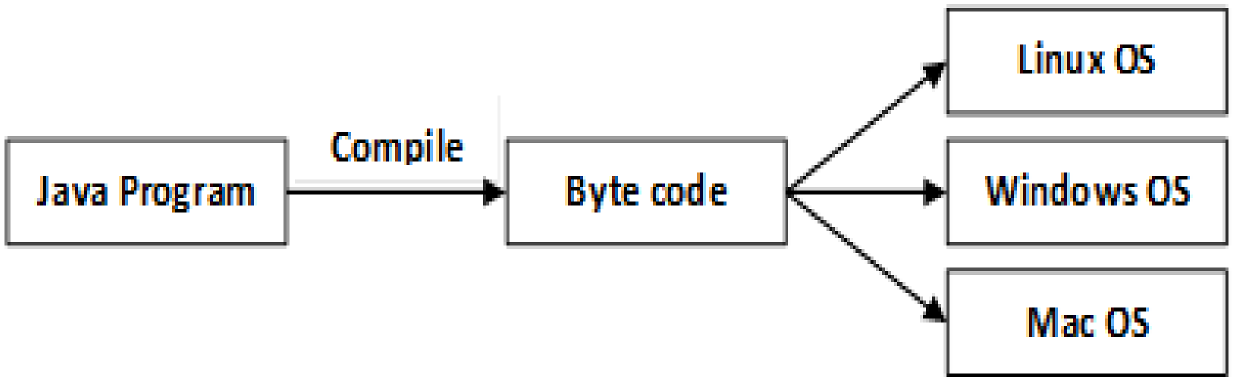
- **Standalone Applications:** Java standalone applications uses GUI components such as AWT, Swing, and Java. These components contain buttons, list, menu, scroll panel, etc. It is also known as desktop alienations.
- **Enterprise Applications:** An application which is distributed in nature is called enterprise applications.
- **Web Applications:** An applications that run on the server is called web applications. We use JSP, Servlet, spring, and Hibernate technologies for creating web applications.
- **Mobile Applications:** Java ME is a cross-platform to develop mobile applications which run across smartphones. Java is a platform for App Development in Android.

JAVA PLATFORM

Java Platform is a collection of programs. It helps to develop and run a program written in the Java programming language. Java Platform includes an execution engine, a compiler and set of libraries. Java is a platform-independent language.

FEATURES OF JAVA

- **Simple:** Java is a simple language because its syntax is simple, clean, and easy to understand. Complex and ambiguous concepts of C++ are either eliminated or re-implemented in Java. For example, pointer and operator overloading are not used in Java.
- **Object-Oriented:** In Java, everything is in the form of the object. It means it has some data and behavior. A program must have at least one class and object.
- **Robust:** Java makes an effort to check error at run time and compile time. It uses a strong memory management system called garbage collector. Exception handling and garbage collection features make it strong.
- **Secure:** Java is a secure programming language because it has no explicit pointer and programs runs in the virtual machine. Java contains a security manager that defines the access of Java classes.
- **Platform-Independent:** Java provides a guarantee that code writes once and run anywhere. This byte code is platform-independent and can be run on any machine.



- **Portable:** Java Byte code can be carried to any platform. No implementation-dependent features. Everything related to storage is predefined, for example, the size of primitive data types.
- **High Performance:** Java is an interpreted language. Java enables high performance with the use of the Just-In-Time compiler.
- **Distributed:** Java also has networking facilities. It is designed for the distributed environment of the internet because it supports TCP/IP protocol. It can run over the internet. EJB and RMI are used to create a distributed system.

- **Multi-threaded:** Java also supports multi-threading. It means to handle more than one job a time.

Java

Our core Java programming tutorial is designed for students and working professionals. Java is an object-oriented, class-based, concurrent, secured and general-purpose computer-programming language. It is a widely used robust technology.

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Java Example

Let's have a quick look at Java programming example. A detailed description of Hello Java example is available in next page.

Simple.java

```
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.

4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

EVENT AND LISTENER (JAVA EVENT HANDLING)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

JAVA EVENT CLASSES AND LISTENER INTERFACES

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

STEPS TO PERFORM EVENT HANDLING

Following steps are required to perform event handling:

1. Register the component with the Listener

REGISTRATION METHODS

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  - public void addActionListener(ActionListener a){}
- **Menu Item**
  - public void addActionListener(ActionListener a){}
- **Text Field**
  - public void addActionListener(ActionListener a){}
  - public void addTextListener(TextListener a){}
- **Text Area**
  - public void addTextListener(TextListener a){}
- **Check box**
  - public void addItemListener(ItemListener a){}
- **Choice**
  - public void addItemListener(ItemListener a){}
- **List**
  - public void addActionListener(ActionListener a){}
  - public void addItemListener(ItemListener a){}

Java Event Handling Code

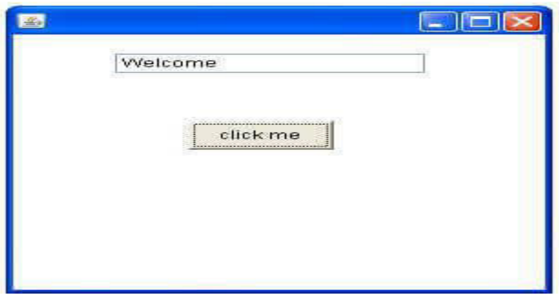
We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Java event handling by implementing Action Listener

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
```

public void set Bounds(int xaxis, int yaxis, int width, int height); have been used in the above example that sets the position of the component it may be button, text field



Java Architecture

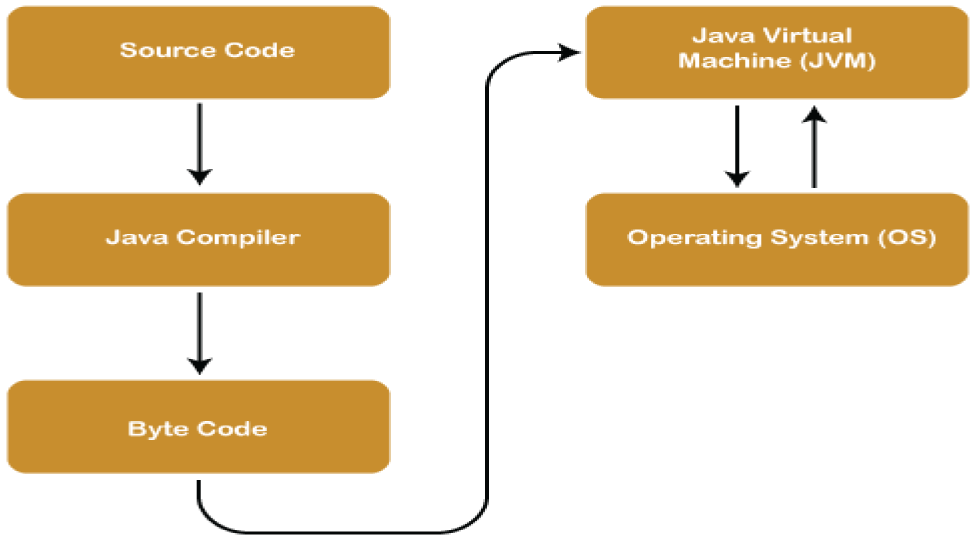
Java Architecture is a collection of components, i.e., **JVM, JRE,** and **JDK**. **It** integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. **Java Architecture** explains each and every step of how a program is compiled and executed.

Java Architecture can be explained by using the following steps:

There is a process of compilation and interpretation in Java.

- Java compiler converts the Java code into byte code.
- After that, the JVM converts the byte code into machine code.
- The machine code is then executed by the machine.

The following figure represents the **Java Architecture** in which each step is elaborate graphically.



Now let's dive deep to get more knowledge about **Java Architecture**. As we know that the Java architecture is a collection of components, so we will discuss each and every component into detail.

Components of Java Architecture

The Java architecture includes the three main components:

- Java Virtual Machine (JVM)
- Java Runtime Environment (JRE)
- Java Development Kit (JDK)

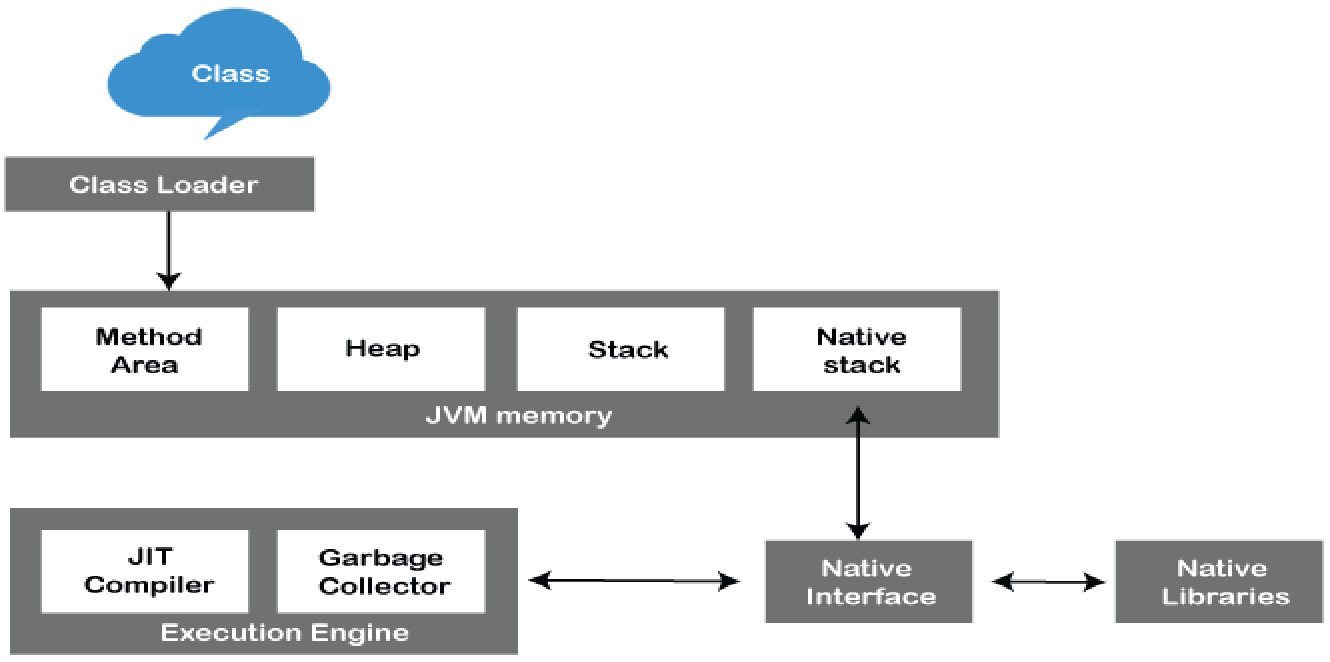
Java Virtual Machine

The main feature of Java is **WORA**. WORA stands for **Write Once Run Anywhere**. The feature states that we can write our code once and use it anywhere or on any operating system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.

JVM, first of all, loads the code into memory and verifies it. After that, it executes the code and provides a runtime environment. Java Virtual Machine (JVM) has its own architecture, which is given below:

JVM Architecture

JVM is an abstract machine that provides the environment in which Java byte code is executed. The falling figure represents the architecture of the JVM.



Class Loader: Class Loader is a subsystem used to load class files. Class Loader first loads the Java code whenever we run it.

Class Method Area: In the memory, there is an area where the class data is stored during the code's execution. Class method area holds the information of static variables, static methods, static blocks, and instance methods.

Heap: The heap area is a part of the JVM memory and is created when the JVM starts up. Its size cannot be static because it increase or decrease during the application runs.

Stack: It is also referred to as thread stack. It is created for a single execution thread. The thread uses this area to store the elements like the partial result, local variable, data used for calling method and returns etc.

Native Stack: It contains the information of all the native methods used in our application.

Execution Engine: It is the central part of the JVM. Its main task is to execute the byte code and execute the Java classes. The execution engine has three main components used for executing Java classes.

- **Interpreter:** It converts the byte code into native code and executes. It sequentially executes the code. The interpreter interprets continuously and even the same method multiple times. This reduces the performance of the system, and to solve this, the JIT compiler is introduced.
- **JIT Compiler:** JIT compiler is introduced to remove the drawback of the interpreter. It increases the speed of execution and improves performance.
- **Garbage Collector:** The garbage collector is used to manage the memory, and it is a program written in Java. It works in two phases, i.e., **Mark** and **Sweep**. Mark is an area where the garbage collector identifies the used and unused chunks of memory. The Sweep removes the identified object from the **Mark**

Java Native Interface

Java Native Interface works as a mediator between Java method calls and native libraries.

Java Runtime Environment

It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it. To learn more about the Java Runtime Environment,

Java Development Kit

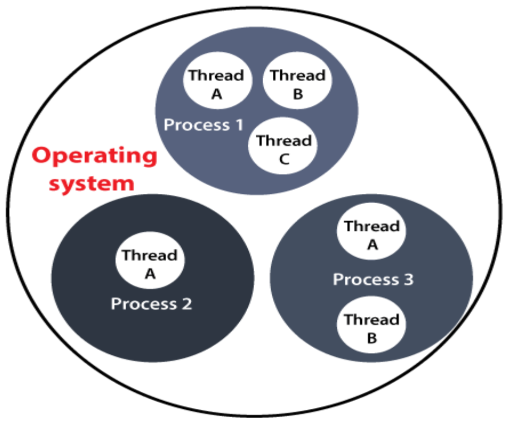
It is a software development environment used in the development of Java applications and applets. Java Development Kit holds JRE, a compiler, an interpreter or loader, and several development tools in it. To learn more about the Java Development Kit, These are three main components of Java Architecture. The execution of a program is done with all these three components.

Thread Concept in Java

Before introducing the **thread concept**, we were unable to run more than one task in parallel. It was a drawback, and to remove that drawback, **Thread Concept** was introduced.

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

In order to perform complicated tasks in the background, we used the **Thread concept in Java**. All the tasks are executed without affecting the main program. In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.



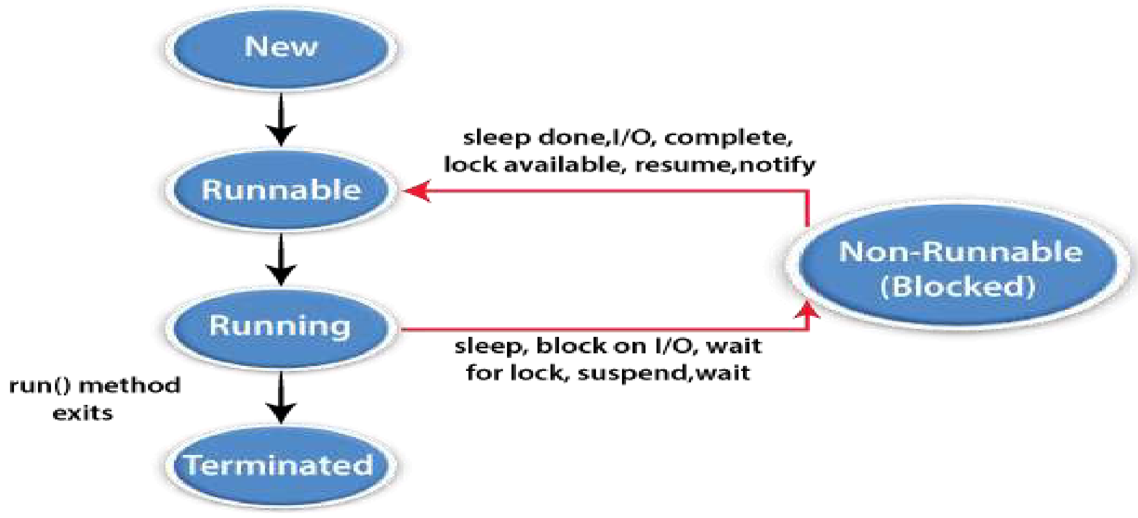
Another benefit of using **thread** is that if a thread gets an exception or an error at the time of its execution, it doesn't affect the execution of the other threads. All the threads share a common memory and have their own stack, local variables and program counter. When multiple threads are executed in parallel at the same time, this process is known as **Multithreading**.

In a simple way, a Thread is a:

- Feature through which we can perform multiple activities within a single process.
- Lightweight process.
- Series of executed statements.
- Nested sequence of method calls.

Thread Model

Just like a process, a thread exists in several states. These states are as follows:


New
Runnable
Running
Terminated
Non-Runnable (Blocked)
sleep done,I/O, complete, lock available, resume,notify
sleep, block on I/O, wait for lock, suspend,wait
run() method exits

1) New (Ready to run)

A thread is in **New** when it gets CPU time.

2) Running

A thread is in **a Running** state when it is under execution.

3) Suspended

A thread is in the **Suspended** state when it is temporarily inactive or under execution.

4) Blocked

A thread is in the **Blocked** state when it is waiting for resources.

5) Terminated

A thread comes in this state when at any given time, it halts its execution immediately.

Creating Thread

A thread is created either by "creating or implementing" the **Runnable Interface** or by extending the **Thread class**. These are the only two ways through which we can create a thread.

Let's dive into details of both these way of creating a thread:

Thread Class

A **Thread class** has several methods and constructors which allow us to perform various operations on a thread. The Thread class extends the **Object** class. The **Object** class implements the **Runnable** interface. The thread class has the following constructors that are used to perform various operations.

- **Thread()**
- **Thread(Runnable, String name)**
- **Thread(Runnable target)**
- **Thread(ThreadGroup group, Runnable target, String name)**
- **Thread(ThreadGroup group, Runnable target)**
- **Thread(ThreadGroup group, String name)**
- **Thread(ThreadGroup group, Runnable target, String name, long stackSize)**

Runnable Interface (run() method)

The Runnable interface is required to be implemented by that class whose instances are intended to be executed by a thread. The runnable interface gives us the **run()** method to perform an action for the thread.

start() method

The method is used for starting a thread that we have newly created. It starts a new thread with a new callstack. After executing the **start()** method, the thread changes the state from New to Runnable. It executes the **run() method** when the thread gets the correct time to execute it.

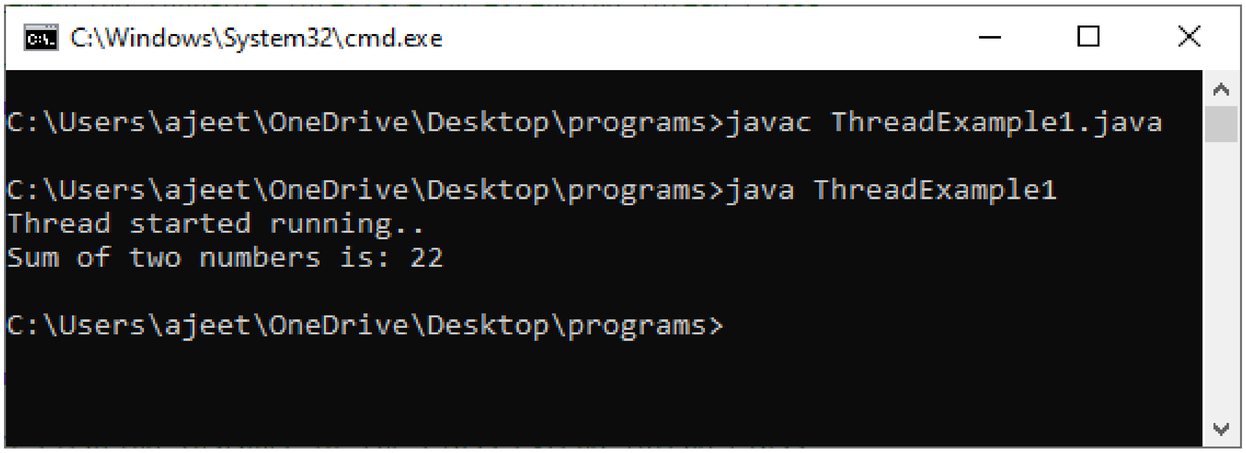
Let's take an example to understand how we can create a Java thread by extending the Thread class:

ThreadExample1.java

```
// Implementing runnable interface by extending Thread class
public class ThreadExample1 extends Thread {
    // run() method to perform action for thread.
    public void run()
    {
      int a= 10;
      int b=12;
      int result = a+b;
      System.out.println("Thread started running..");
      System.out.println("Sum of two numbers is: "+ result);
```

```
        }
        public static void main( String args[] )
        {
         // Creating instance of the class extend Thread class
           ThreadExample1 t1 = new  ThreadExample1();
           //calling start method to execute the run() method of the Thread class
           t1.start();
        }
    }
```

Output:



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac ThreadExample1.java

C:\Users\ajeet\OneDrive\Desktop\programs>java ThreadExample1
Thread started running..
Sum of two numbers is: 22

C:\Users\ajeet\OneDrive\Desktop\programs>
```

Creating thread by implementing the runnable interface

In Java, we can also create a thread by implementing the runnable interface. The runnable interface provides us both the run() method and the start() method.

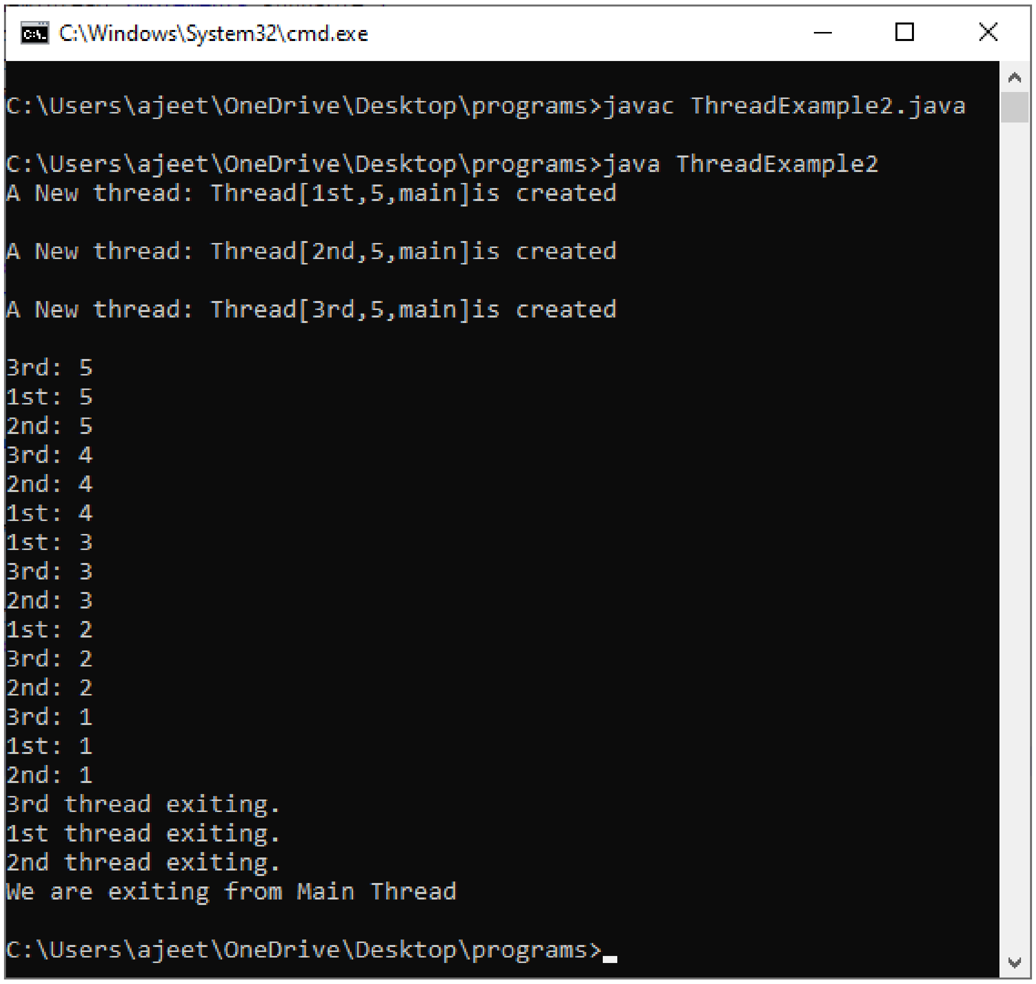
Let's takes an example to understand how we can create, start and run the thread using the runnable interface.

Let's takes an example to understand how we can create, start and run the thread using the runnable interface.

ThreadExample2.java

```java
class NewThread implements Runnable {
   String name;
   Thread thread;
   NewThread (String name){
      this.name = name;
      thread = new Thread(this, name);
      System.out.println( "A New thread: " + thread+ "is created\n" );
      thread.start();
   }
   public void run() {
   try {
      for(int j = 5; j > 0; j--) {
         System.out.println(name + ": " + j);
         Thread.sleep(1000);
      }
   }catch (InterruptedException e) {
      System.out.println(name + " thread Interrupted");
   }
    System.out.println(name + " thread exiting.");
   }
}
class ThreadExample2 {
   public static void main(String args[]) {
      new NewThread("1st");
      new NewThread("2nd");
      new NewThread("3rd");
      try {
         Thread.sleep(8000);
      } catch (InterruptedException excetion) {
         System.out.println("Inturruption occurs in Main Thread");
      }
      System.out.println("We are exiting from Main Thread");
   }
}
```

Output:



```
C:\Users\ajeet\OneDrive\Desktop\programs>javac ThreadExample2.java

C:\Users\ajeet\OneDrive\Desktop\programs>java ThreadExample2
A New thread: Thread[1st,5,main]is created

A New thread: Thread[2nd,5,main]is created

A New thread: Thread[3rd,5,main]is created

3rd: 5
1st: 5
2nd: 5
3rd: 4
2nd: 4
1st: 4
1st: 3
3rd: 3
2nd: 3
1st: 2
3rd: 2
2nd: 2
3rd: 1
1st: 1
2nd: 1
3rd thread exiting.
1st thread exiting.
2nd thread exiting.
We are exiting from Main Thread

C:\Users\ajeet\OneDrive\Desktop\programs>_
```

<u>Java Networking</u>

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an Ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

Java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:

Authenticator	Cache Request
Cache Response	Content Handler
Cookie Handler	Cookie Manager
Data gram Packet	Data gram Socket
Data gram Socket Impl	Interface Address
Jar URL Connection	Multicast Socket
Inet Socket Address	Inet Address
Inet4Address	Inet6Address

List of interfaces available in java.net package:

Content Handler Factory	Cookie Policy
Cookie Store	Datagram Socket Impl Factory
File Name Map	Socket Option<T>
Socket Options	Socket Impl Factory
URL Stream Handler Factory	Protocol Family

NETWORKING FEATURES

In Java, there are several networking features that allow developers to create networked applications. Some of the most commonly used networking features in Java are:

Java.net package:

This package provides classes to implement network protocols, such as TCP/IP, UDP, Sockets, and URLs. It allows developers to establish connections and communicate with other computers over a network.

Socket programming:

Java's java.net.Socket class provides a low-level interface for networking. It allows developers to create client-server applications using TCP/IP or UDP protocols. Sockets can be used to send and receive data over a network.

URL handling:

Java's java.net.URL class provides methods to handle URLs. It allows developers to read and write data from a specified URL, such as reading a webpage or downloading a file from the internet.

InetAddress class:

The java.net.InetAddress class is used to represent Internet Protocol (IP) addresses. It provides methods to get the IP address of a host, resolve hostnames, and perform IP address manipulation.

Network interfaces:

Java's java.net.NetworkInterface class provides methods to retrieve information about network interfaces on the system. It allows developers to determine available network interfaces, their IP addresses, and other related information.

Server socket class:

Java's java.net.ServerSocket class allows developers to create server applications that listen for incoming connections on a specific port. This class is used to implement server-side socket programming.

Datagram Socket and Datagram Packet classes:

These classes are used for UDP-based networking in Java. The java.net.DatagramSocket class represents a connectionless socket for sending and receiving datagram packets. The java.net.DatagramPacket class encapsulates the data to be sent or received, along with the destination or source address.

Multicast Socket class:

The java.net.MulticastSocket class is used to implement multicast networking in Java. It allows sending and receiving IP multicast packets, enabling communication between multiple hosts on a network.

These are just a few of the networking features provided by Java. By utilizing these features, developers can create robust and efficient networked applications in Java.

MULTIMEDIA

Multimedia is an engaging kind of media that offers a variety of effective ways to convey information to users. Users can interact with digital information through it. It serves as a communication tool. Education, training, reference materials, corporate presentations, marketing, and documentary are a few industries that heavily utilize multimedia.

Multimedia, by definition, is the use of text, audio, video, graphics, and animation to convey information in an engaging and dynamic way. In other terms, multimedia is a technological way of presenting information that combines audio, video, images, and animations with textual data. Examples include video conferencing, Yahoo Messenger, email, and the Multimedia Messaging Service (MMS Service (MMS).

As the name implies, multimedia is the combination of the words "multi" and "media," which refers to the various media (hardware/software) utilized for information transmission.

COMPONENTS OF MULTIMEDIA

The following are typical multimedia elements:

1) Text - Text appears in all multi-media projects to some extent. To match the successful presentation of the multimedia program, the text may be presented in a variety of font styles and sizes.

2) Graphics - The multimedia program is appealing because of its graphics. People frequently find it difficult to read long passages of text on screens. As a result, visuals are frequently utilized instead of writing to convey ideas, give context, etc. Graphics can be of two different types:

- **Bitmap -** Bitmap images are authentic pictures that can be taken using tools like digital cameras or scanners. Bitmap pictures are often not modifiable. Memory use for bitmap pictures is high.
- **Vector Graphics -** Computers can draw vector graphics because they just need a little amount of memory. These images can be changed.

3) Animation - A static picture can be animated to appear to be in motion. A continuous succession of static images shown in order is all that makes up an animation. Effective attention-getting may be achieved by the animation. Additionally, animation adds levity and appeal to a presentation. In multimedia applications, the animation is fairly common.

4) Audio - Speech, music, and sound effects could all be necessary for a multimedia application. They are referred to as the audio or sound component of multimedia. Speaking is a fantastic educational tool. Analog and digital audio are both kinds. The initial sound signal is referred to as analog audio or sound. Digital sound is saved on a computer. Digital audio is therefore utilized for sound in multimedia applications.

5) Video - The term "video" describes a moving image that is supported by sound, such as a television image. A multimedia application's video component conveys a lot of information quickly. For displaying real-world items in multimedia applications, digital video is helpful. If uploaded to the internet, the video really does have the highest performance requirements for computer memory and bandwidth. The quality of digital video files may still be preserved while being saved on a computer, similarly to other data. A computer network allows for the transport of digital video files. The digital video snippets are simple to modify.

APPLICATIONS OF MULTIMEDIA

The typical areas where multimedia is applied are listed below.

1) For entertainment purposes - Multimedia marketing may significantly improve the promotion of new items. Both advertising and marketing staff had their doors opened by the economical communication boost provided by multimedia. Flying banner presentations, video transitions, animations, and audio effects are just a few of the components utilized to create a multimedia-based advertisement that appeals to the customer in a brand-new way and encourages the purchase of the goods.

2) For education purposes - There are currently a lot of educational computer games accessible. Take a look at an illustration of an educational app that plays children's rhymes. In addition to merely repeating rhymes, the youngster may create drawings, scale items up or down, and more. There are many more multimedia products on the market that provide children with a wealth of in-depth knowledge and playing options.

3) For business purposes - There are several commercial uses for multimedia. Multimedia and communication technologies have made it possible for information from international work groups. Today's team members can work remotely and for a variety of businesses. A global workplace will result from this. The following facilities should be supported by the multimedia network:

- Office needs
- Records management
- Employee training
- Electronic mail
- Voice mail

4) For marketing purposes - Multimedia marketing may significantly improve the promotion of new items. Both advertising and promotion staff had their doors opened by the economical communication boost provided by multimedia. Flying banner presentations, video transitions, animations, and audio effects are just a few of the components utilized to create a multimedia-based advertisement that appeals to the customer in a brand-new way and encourages the purchase of the goods.

5) For banking purposes - Another public setting where multimedia is being used more and more recently is banks. People visit banks to open savings and current accounts, make deposits and withdrawals, learn about the bank's various financial plans, apply for loans, and other things. Each bank wants to notify its consumers with a wealth of information. It can employ multimedia in a variety of ways to do this. The bank also has a PC monitor in the clients' rest area that shows details about its numerous programs. Online and internet banking have grown in popularity recently. These heavily rely on multimedia. As a result, banks are using multimedia to better serve their clients and inform them of their appealing financing options.

There are several media techniques that can be used in Java programming:

Image Manipulation:
Java provides various libraries and classes for loading, manipulating, and saving images. For example, the javax.imageio package allows developers to read and write images in different formats, while java.awt.image package offers classes for performing operations like scaling, cropping, and filtering on images.

Audio Playback:
Java Sound API enables developers to play audio files of different formats. It provides classes like Clip and AudioInputStream to load and manipulate audio data. JavaFX also provides media classes for playing audio files, along with advanced features such as playback controls and volume management.

Video Playback:
JavaFX has built-in media capabilities, which allow developers to play video files in different formats. The javafx.scene.media package provides classes like MediaPlayer and MediaView for handling video playback. By using these classes, you can control video playback, handle events, and even apply visual effects to the video.

Animation:
Java provides several libraries and frameworks that support animation. For example, JavaFX provides a powerful animation framework with classes like Animation, Timeline, and KeyFrame. These classes allow you to create smooth animations by specifying time-based actions and transitions.

3D Graphics:
Java 3D API provides support for creating and manipulating 3D graphics in Java. It offers a comprehensive set of classes and utilities for constructing and rendering 3D scenes. With Java 3D, you can create complex 3D objects, apply textures and lighting effects, and even perform animations in three-dimensional space.

Web Content Integration:
JavaFX provides WebView class that enables developers to embed web content in Java applications. This allows you to integrate media elements like images, audio, and video from web sources into your Java application.

These are just a few examples of media techniques that can be used in Java programming. Java provides a rich ecosystem of libraries and frameworks, making it a versatile language for handling various types of media.

There are several media techniques that can be used in Java programming:

Image Manipulation:

Java provides various libraries and classes for loading, manipulating, and saving images. For example, the javax.imageio package allows developers to read and write images in different formats, while java.awt.image package offers classes for performing operations like scaling, cropping, and filtering on images.

Audio Playback:

Java Sound API enables developers to play audio files of different formats. It provides classes like Clip and AudioInputStream to load and manipulate audio data. JavaFX also provides media classes for playing audio files, along with advanced features such as playback controls and volume management.

Video Playback:

JavaFX has built-in media capabilities, which allow developers to play video files in different formats. The javafx.scene.media package provides classes like MediaPlayer and MediaView for handling video playback. By using these classes, you can control video playback, handle events, and even apply visual effects to the video.

Animation:

Java provides several libraries and frameworks that support animation. For example, JavaFX provides a powerful animation framework with classes like Animation, Timeline, and KeyFrame. These classes allow you to create smooth animations by specifying time-based actions and transitions.

3D Graphics:

Java 3D API provides support for creating and manipulating 3D graphics in Java. It offers a comprehensive set of classes and utilities for constructing and rendering 3D scenes. With Java 3D, you can create complex 3D objects, apply textures and lighting effects, and even perform animations in three-dimensional space.

Web Content Integration:

JavaFX provides WebView class that enables developers to embed web content in Java applications. This allows you to integrate media elements like images, audio, and video from web sources into your Java application.

These are just a few examples of media techniques that can be used in Java programming. Java provides a rich ecosystem of libraries and frameworks, making it a versatile language for handling various types of media.

UNIT-2
REMOTE METHOD INVOCATION

RMI (Remote Method Invocation)

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

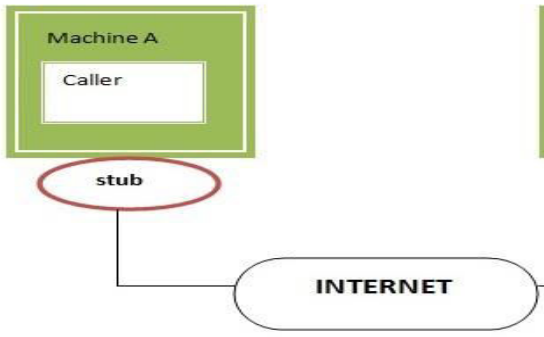
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for



skeletons.

Understanding requirements for the distributed applications

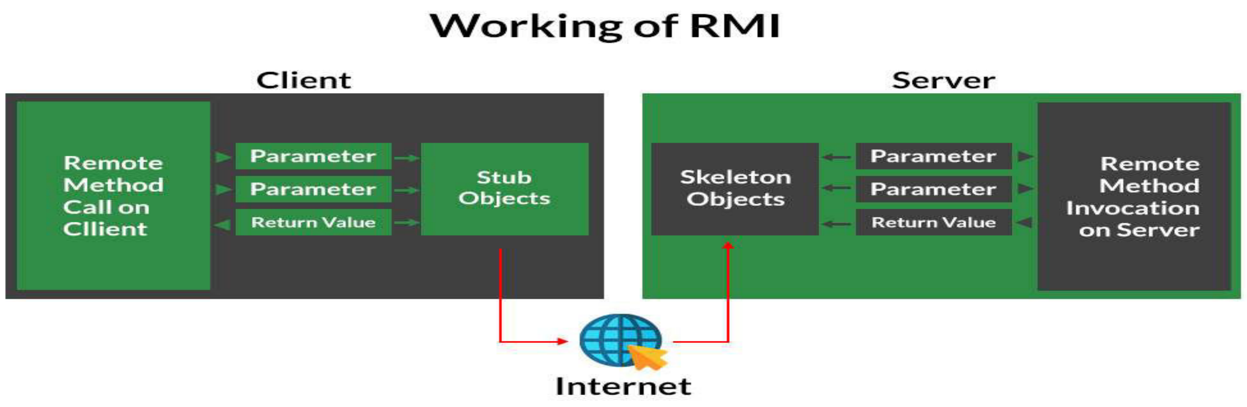
If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Working of RMI

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:

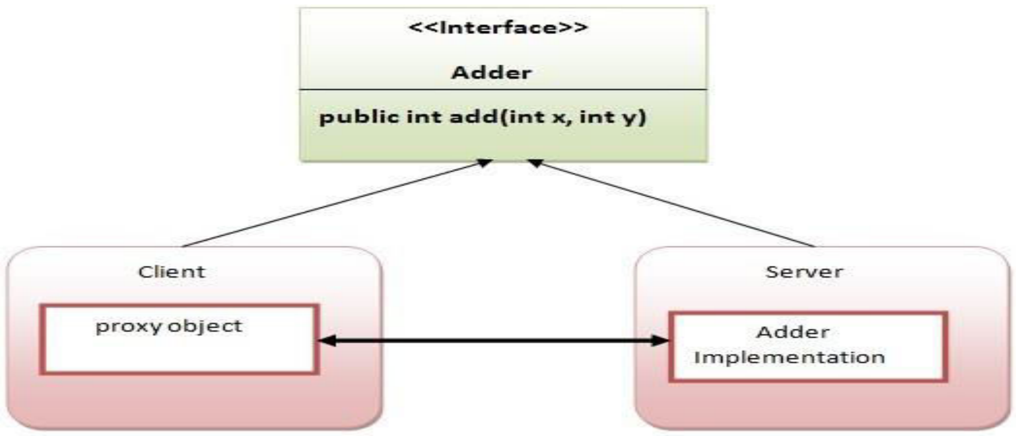


These are the steps to be followed sequentially to implement Interface as defined below as follows:

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool

5. Create and start the remote application
6. Create and start the client application

The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



1) Create the remote interface

For creating the remote interface, extend the Remote interface and declare the Remote Exception with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares Remote Exception.

```
import java.rmi.*;
public interface Adder extends Remote{
public int add(int x,int y)throws Remote Exception;
}
```

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares Remote Exception.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) Create the stub and skeleton objects using the rmic tool.

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic Adder Remote
```

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

```
rmiregistry 5000
```

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It returns the reference of the remote object.
public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;	It binds the remote object with the given name.
public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;	It destroys the remote object which is bound with the given name.
public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;	It binds the remote object to the new name.
public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;	It returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name soon.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```java
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

DISTRIBUTED APPLICATION ARCHITECTURE

Distributed application architecture is a design approach for developing software applications that are comprised of multiple components or services that are deployed and run on multiple computers or servers. This architecture allows for scalability, fault-tolerance, and easy integration of different components

There are several key aspects of distributed application architecture:

Client-server model:

The application is divided into a client component that interacts with the user and a server component that provides services and resources. Clients communicate with servers over a network, typically using protocols like HTTP or TCP/IP.

Middleware:

Middleware is a software layer that sits between the client and server components, providing services such as message passing, data persistence, and load balancing. It coordinates communication between different components and facilitates interoperability.

Service-oriented architecture (SOA):

This architectural style focuses on organizing software applications into services, which are self-contained modules that provide specific functions or processes. Services can be developed and deployed independently, and can communicate with each other to form a complete application.

Micro services architecture:

This approach breaks down the application into smaller, loosely coupled services that can be developed, deployed, and scaled independently. Each microservice is responsible for a specific functionality and can communicate with each other through lightweight APIs.

Message-based communication:

In a distributed application, components typically communicate with each other through messages. This can be achieved using messaging protocols like AMQP or event-driven architectures, where components publish and subscribe to events.

Load balancing and fault tolerance:

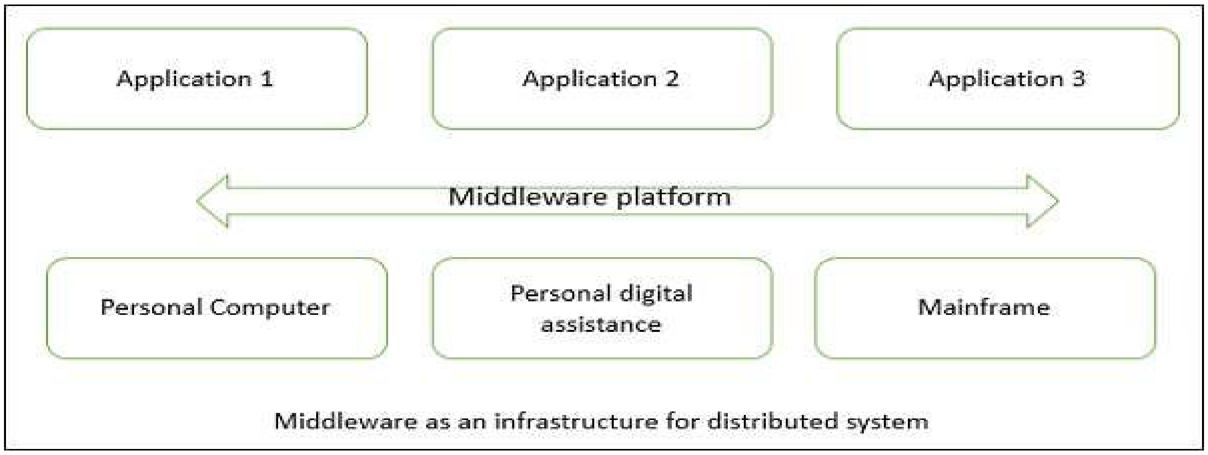
Distributed applications often involve multiple instances of a component running on different servers. Load balancing ensures that requests are distributed evenly across these instances, improving performance and scalability. Fault tolerance mechanisms like redundancy and failover ensure that the application can continue functioning even if one or more components fail.

Overall, distributed application architecture aims to provide scalable, reliable, and flexible software solutions by distributing components across multiple servers and leveraging communication protocols and middleware to enable seamless integration and coordination.

In distributed architecture, components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal.

- In this architecture, information processing is not confined to a single machine rather it is distributed over several independent computers.
- A distributed system can be demonstrated by the client-server architecture which forms the base for multi-tier architectures; alternatives are the broker architecture such as CORBA, and the Service-Oriented Architecture (SOA).
- There are several technology frameworks to support distributed architectures, including .NET, J2EE, CORBA, .NET Web services, AXIS Java Web services, and Globus Grid services.
- Middleware is an infrastructure that appropriately supports the development and execution of distributed applications. It provides a buffer between the applications and the network.
- It sits in the middle of system and manages or supports the different components of a distributed system. Examples are transaction processing monitors, data convertors and communication controllers etc.

Middleware as an infrastructure for distributed system



The basis of a distributed architecture is its transparency, reliability, and availability.

The following table lists the different forms of transparency in a distributed system −

Transparency & Description

1. **Access** - Hides the way in which resources are accessed and the differences in data platform.
2. **Location** - Hides where resources are located.
3. **Technology -** Hides different technologies such as programming language and OS from user.
4. **Migration / Relocation** - Hide resources that may be moved to another location which are in use.
5. **Replication** - Hide resources that may be copied at several location.
6. **Concurrency** - Hide resources that may be shared with other users.
7. **Failure** **- Hides** failure and recovery of resources from user.
8. **Persistence** - Hides whether a resource (software) is in memory or disk.

Advantages

- **Resource sharing** − Sharing of hardware and software resources.
- **Openness** − Flexibility of using hardware and software of different vendors.
- **Concurrency** − Concurrent processing to enhance performance.
- **Scalability** − Increased throughput by adding new resources.
- **Fault tolerance** − The ability to continue in operation after a fault has occurred.

Disadvantages

- **Complexity** − They are more complex than centralized systems.
- **Security** − More susceptible to external attack.
- **Manageability** − More effort required for system management.

- **Unpredictability** − Unpredictable responses depending on the system organization and network load.

Centralized System vs. Distributed System

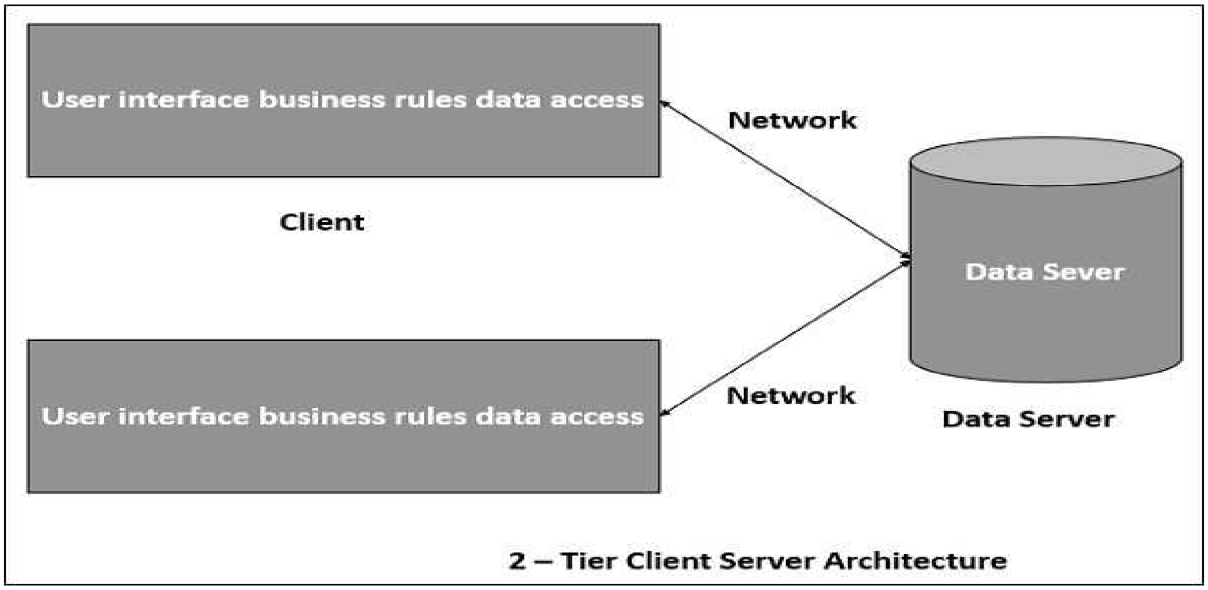
Criteria	Centralized system	Distributed System
Economics	Low	High
Availability	Low	High
Complexity	Low	High
Consistency	Simple	High
Scalability	Poor	Good
Technology	Homogeneous	Heterogeneous
Security	High	Low

Client-Server Architecture

The client-server architecture is the most common distributed system architecture which decomposes the system into two major subsystems or logical processes −

- **Client** − this is the first process that issues a request to the second process i.e. the server.
- **Server** − this is the second process that receives the request, carries it out, and sends a reply to the client.

In this architecture, the application is modelled as a set of services that are provided by servers and a set of clients that use these services. The servers need not know about clients, but the clients must know the identity of servers, and the mapping of processors to processes is not necessarily 1: 1


User interface business rules data access
Network
Client
Data Sever
User interface business rules data access
Network
Data Server
2 – Tier Client Server Architecture

Client-server Architecture can be classified into two models based on the functionality of the client −

Thin-client model

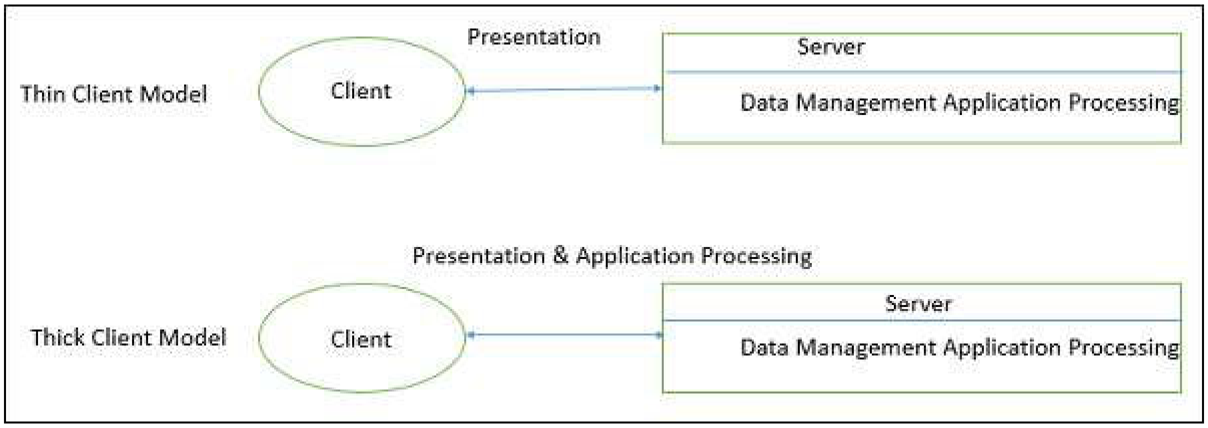
In thin-client model, all the application processing and data management is carried by the server. The client is simply responsible for running the presentation software.

- Used when legacy systems are migrated to client server architectures in which legacy system acts as a server in its own right with a graphical interface implemented on a client
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Thick/Fat-client model

In thick-client model, the server is only in charge for data management. The software on the client implements the application logic and the interactions with the system user.

- Most appropriate for new C/S systems where the capabilities of the client system are known in advance
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.


Thin Client Model
Client
Presentation
Server
Data Management Application Processing
Thick Client Model
Client
Presentation & Application Processing
Server
Data Management Application Processing

Advantages

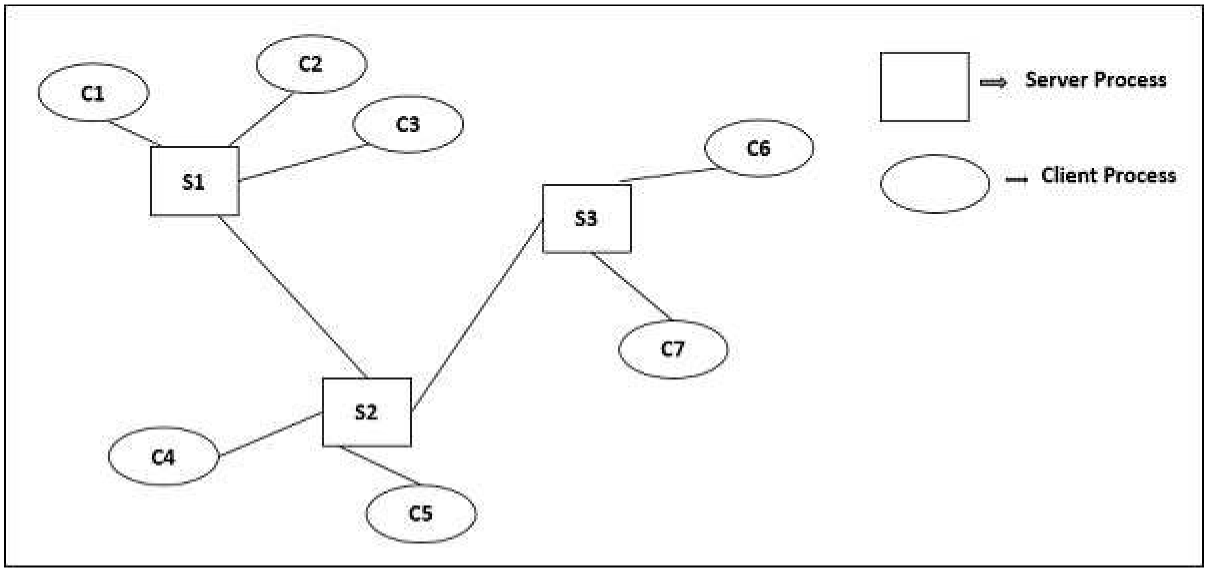
- Separation of responsibilities such as user interface presentation and business logic processing.
- Reusability of server components and potential for concurrency
- Simplifies the design and the development of distributed applications
- It makes it easy to migrate or integrate existing applications into a distributed environment.
- It also makes effective use of resources when a large number of clients are accessing a high-performance server.

Disadvantages

- Lack of heterogeneous infrastructure to deal with the requirement changes.
- Security complications.
- Limited server availability and reliability.
- Limited testability and scalability.
- Fat clients with presentation and business logic together.

Multi-Tier Architecture (n-tier Architecture)

Multi-tier architecture is a client–server architecture in which the functions such as presentation, application processing, and data management are physically separated. By separating an application into tiers, developers obtain the option of changing or adding a specific layer, instead of reworking the entire application. It provides a model by which developers can create flexible and reusable applications.



The most general use of multi-tier architecture is the three-tier architecture. A three-tier architecture is typically composed of a presentation tier, an application tier, and a data storage tier and may execute on a separate processor.

Presentation Tier

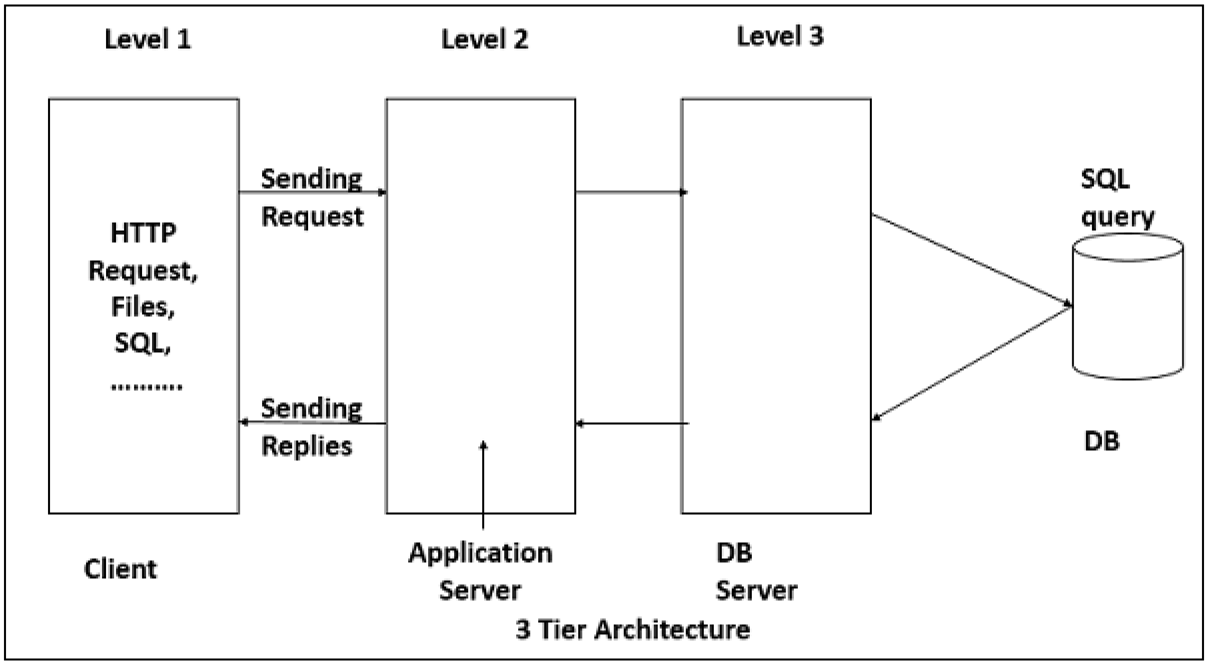
Presentation layer is the topmost level of the application by which users can access directly such as webpage or Operating System GUI (Graphical User interface). The primary function of this layer is to translate the tasks and results to something that user can understand. It communicates with other tiers so that it places the results to the browser/client tier and all other tiers in the network.

Application Tier (Business Logic, Logic Tier, or Middle Tier)

Application tier coordinates the application, processes the commands, makes logical decisions, evaluation, and performs calculations. It controls an application's functionality by performing detailed processing. It also moves and processes data between the two surrounding layers.

Data Tier

In this layer, information is stored and retrieved from the database or file system. The information is then passed back for processing and then back to the user. It includes the data persistence mechanisms (database servers, file shares, etc.) and provides API (Application Programming Interface) to the application tier which provides methods of managing the stored data.



Advantages

- Better performance than a thin-client approach and is simpler to manage than a thick-client approach.
- Enhances the reusability and scalability − as demands increase, extra servers can be added.
- Provides multi-threading support and also reduces network traffic.
- Provides maintainability and flexibility

Disadvantages

- Unsatisfactory Testability due to lack of testing tools.
- More critical server reliability and availability.

Broker Architectural Style

Broker Architectural Style is a middleware architecture used in distributed computing to coordinate and enable the communication between registered servers and clients. Here, object communication takes place through a middleware system called an object request broker (software bus).

- Client and the server do not interact with each other directly. Client and server have a direct connection to its proxy which communicates with the mediator-broker.
- A server provides services by registering and publishing their interfaces with the broker and clients can request the services from the broker statically or dynamically by look-up.
- CORBA (Common Object Request Broker Architecture) is a good implementation example of the broker architecture.

Components of Broker Architectural Style

The components of broker architectural style are discussed through following heads −

Broker

Broker is responsible for coordinating communication, such as forwarding and dispatching the results and exceptions. It can be either an invocation-oriented service, a document or message - oriented broker to which clients send a message.

- It is responsible for brokering the service requests, locating a proper server, transmitting requests, and sending responses back to clients.
- It retains the servers' registration information including their functionality and services as well as location information.
- It provides APIs for clients to request, servers to respond, registering or unregistering server components, transferring messages, and locating servers.

Stub

Stubs are generated at the static compilation time and then deployed to the client side which is used as a proxy for the client. Client-side proxy acts as a mediator between the client and the broker and provides additional transparency between them and the client; a remote object appears like a local one.

The proxy hides the IPC (inter-process communication) at protocol level and performs marshaling of parameter values and un-marshaling of results from the server.

Skeleton

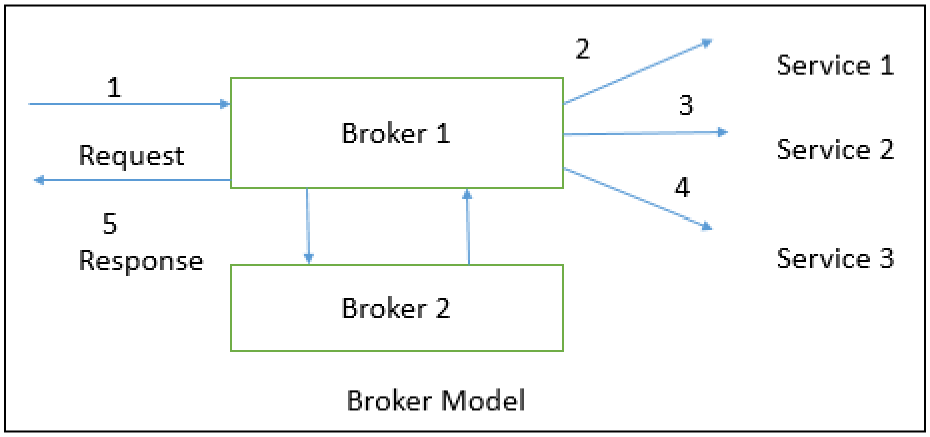
Skeleton is generated by the service interface compilation and then deployed to the server side, which is used as a proxy for the server. Server-side proxy encapsulates low-level system-specific networking functions and provides high-level APIs to mediate between the server and the broker.

It receives the requests, unpacks the requests, unmarshals the method arguments, calls the suitable service, and also marshals the result before sending it back to the client.

Bridge

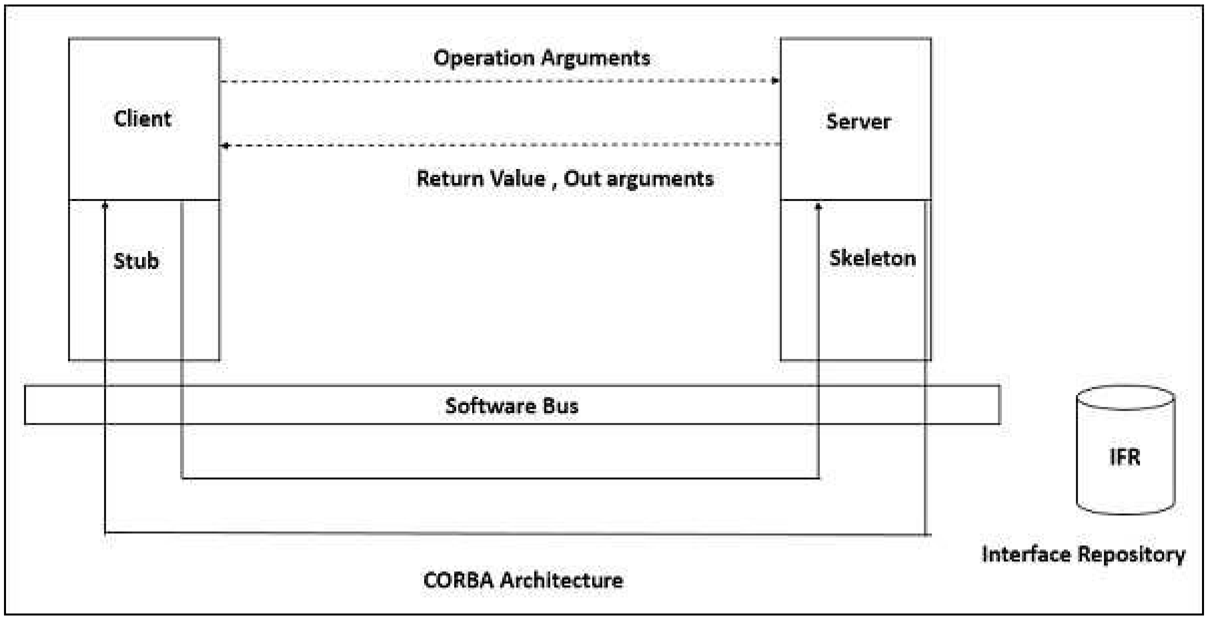
A bridge can connect two different networks based on different communication protocols. It mediates different brokers including DCOM, .NET remote, and Java CORBA brokers.

Bridges are optional component, which hides the implementation details when two brokers interoperate and take requests and parameters in one format and translate them to another format.



Broker implementation in CORBA

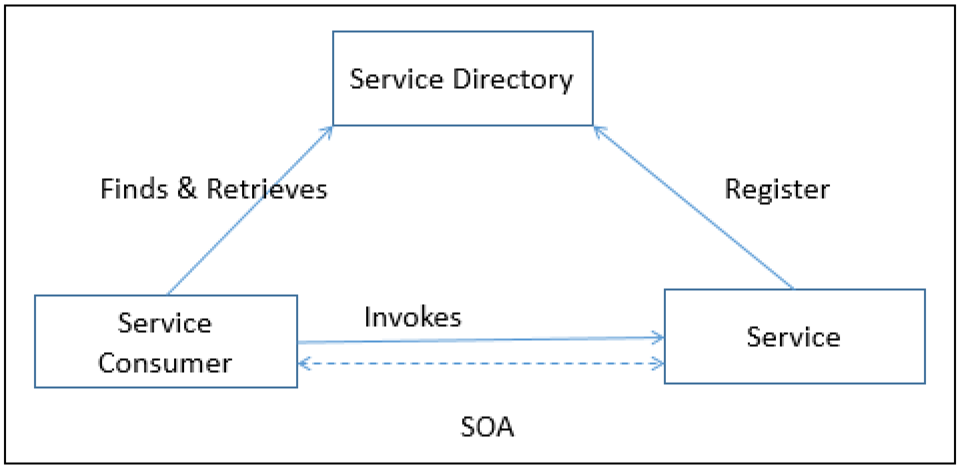
CORBA is an international standard for an Object Request Broker – a middleware to manage communications among distributed objects defined by OMG (object management group).



Service-Oriented Architecture (SOA)

A service is a component of business functionality that is well-defined, self-contained, independent, published, and available to be used via a standard programming interface. The connections between services are conducted by common and universal message-oriented protocols such as the SOAP Web service protocol, which can deliver requests and responses between services loosely.

Service-oriented architecture is a client/server design which support business-driven IT approach in which an application consists of software services and software service consumers (also known as clients or service requesters).



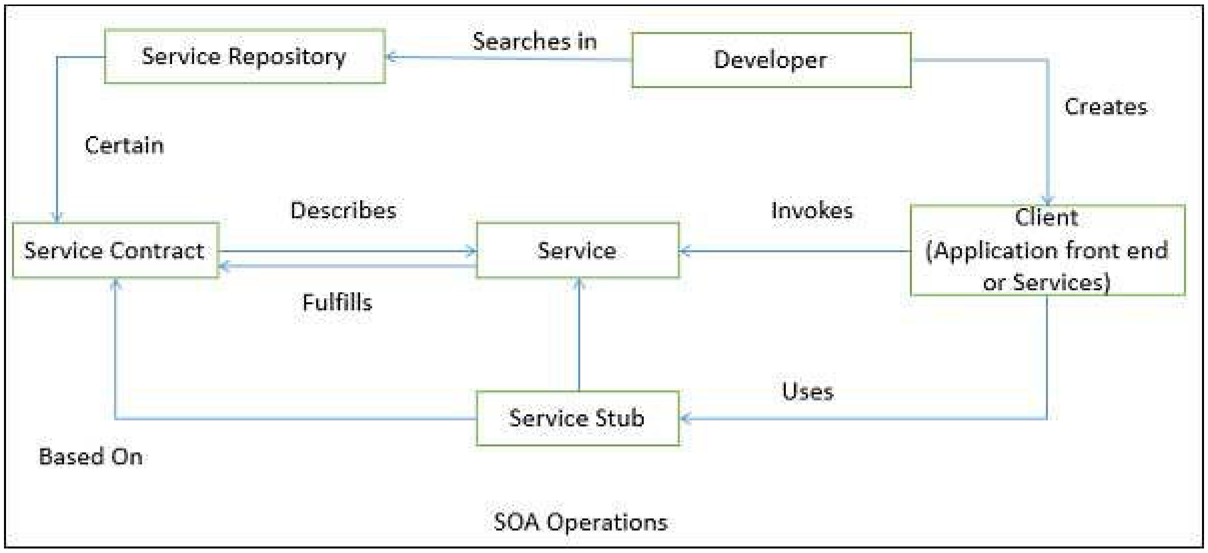
Features of SOA

A service-oriented architecture provides the following features −

- **Distributed Deployment** − Expose enterprise data and business logic as loosely, coupled, discoverable, structured, standard-based, coarse-grained, stateless units of functionality called services.
- **Composability** − Assemble new processes from existing services that are exposed at a desired granularity through well defined, published, and standard complaint interfaces.
- **Interoperability** − Share capabilities and reuse shared services across a network irrespective of underlying protocols or implementation technology.
- **Reusability** − Choose a service provider and access to existing resources exposed as services.

SOA Operation

The following figure illustrates how does SOA operate −



Advantages

- Loose coupling of service–orientation provides great flexibility for enterprises to make use of all available service recourses irrespective of platform and technology restrictions.
- Each service component is independent from other services due to the stateless service feature.
- The implementation of a service will not affect the application of the service as long as the exposed interface is not changed.
- A client or any service can access other services regardless of their platform, technology, vendors, or language implementations.

- Reusability of assets and services since clients of a service only need to know its public interfaces, service composition.
- SOA based business application development are much more efficient in terms of time and cost.
- Enhances the scalability and provide standard connection between systems.
- Efficient and effective usage of 'Business Services'.
- Integration becomes much easier and improved intrinsic interoperability.
- Abstract complexity for developers and energize business processes closer to end users.

Stub/Skeleton Interfaces

This section contains the interfaces and classes used by the stubs and skeletons generated by the rmic stub compiler.

The Remote Stub Class

The java.rmi.server.RemoteStub class is the common superclass for stubs of remote objects. Stub objects are surrogates that support exactly the same set of remote interfaces defined by the actual implementation of a remote object.

```
package java.rmi.server;

public abstract class RemoteStub extends java.rmi.RemoteObject {
        protected RemoteStub() {...}
        protected RemoteStub(RemoteRef ref) {...}

        protected static void setRef(RemoteStub stub, RemoteRef ref) {...}
}
```

The first constructor of RemoteStub creates a stub with a null remote reference. The second constructor creates a stub with the given remote reference, *ref*.

The setRef method is deprecated (and unsupported) as of the Java 2 SDK, Standard Edition, v1.2.

Type Equivalency of Remote Objects with a Stub Class

Clients interact with stub (surrogate) objects that have *exactly* the same set of remote interfaces defined by the remote object's class; the stub class does not include the non-remote portions of the class hierarchy that constitutes the object's type graph. This is because the stub class is generated from the most refined implementation class that implements one or more remote interfaces. For example, if C extends B and B extends A, but only B implements a remote interface, then a stub is generated from B, not C.

Because the stub implements the same set of remote interfaces as the remote object's class, the stub has the same type as the remote portions of the server object's type graph. A client, therefore, can make use of the built-in Java programming language operations to check a remote object's type and to cast from one remote interface to another.

Stubs are generated using the rmic compiler.

The Semantics of Object Methods Declared final

The following methods are declared final in the java.lang.Object class and therefore cannot be overridden by any implementation:

- getClass
- notify
- notifyAll
- wait

The default implementation for getClass is appropriate for all objects written in the Java programming language, local or remote; so, the method needs no special implementation for remote objects. When used on a remote stub, the getClass method reports the exact type of the stub object, generated by rmic. Note that stub type reflects only the remote interfaces implemented by the remote object, not that object's local interfaces.

The wait and notify methods of java.lang.Object deal with waiting and notification in the context of the Java programming language's threading model. While use of these methods for remote stubs does not break the threading model, these methods do not have the same semantics as they do for local objects written in the Java programming language. Specifically, these methods operate on the client's local reference to the remote object (the stub), not the actual object at the remote site.

The Remote Call Interface

The interface RemoteCall is an abstraction used by the stubs and skeletons of remote objects to carry out a call to a remote object.

Note: The RemoteCall interface is deprecated as of the Java 2 SDK, Standard Edition, v1.2. The 1.2 stub protocol does not make use of this interface anymore. As of the Java 2 SDK, Standard Edition, v1.2, stubs now use the new invoke method which does not require Remote Call as a parameter.

```
package java.rmi.server;

import java.io.*;
```

```
public interface RemoteCall {
	ObjectOutput getOutputStream() throws IOException;
	void releaseOutputStream() throws IOException;
	ObjectInput getInputStream() throws IOException;
	void releaseInputStream() throws IOException;
	ObjectOutput getResultStream(boolean success)
		throws IOException, StreamCorruptedException;
	void executeCall() throws Exception;
	void done() throws IOException;
}
```

The method getOutputStream returns the output stream into which either the stub marshals arguments or the skeleton marshals results.

The method releaseOutputStream releases the output stream; in some transports this will release the stream.

The method getInputStream returns the InputStream from which the stub unmarshals results or the skeleton unmarshals parameters.

The method releaseInputStream releases the input stream. This will allow some transports to release the input side of a connection early.

The method getResultStream returns an output stream (after writing out header information relating to the success of the call). Obtaining a result stream should only succeed once per remote call. If *success* is true, then the result to be marshaled is a normal return; otherwise the result is an exception. StreamCorruptedException is thrown if the result stream has already been obtained for this remote call.

The method executeCall does whatever it takes to execute the call.

The method done allows cleanup after the remote call has completed.

The RemoteRef Interface

The interface RemoteRef represents the handle for a remote object. Each stub contains an instance of RemoteRef that contains the concrete representation of a reference. This remote reference is used to carry out remote calls on the remote object for which it is a reference.

```
package java.rmi.server;

public interface RemoteRef extends java.io.Externalizable {
        Object invoke(Remote obj,
                java.lang.reflect.Method method,
                Object[] params,
                long opnum)
            throws Exception;

        RemoteCall newCall(RemoteObject obj, Operation[] op, int opnum,
                    long hash) throws RemoteException;
        void invoke(RemoteCall call) throws Exception;
        void done(RemoteCall call) throws RemoteException;
        String getRefClass(java.io.ObjectOutput out);
        int remoteHashCode();
        boolean remoteEquals(RemoteRef obj);
        String remoteToString();
    }
```

The invoke(Remote,Method,Object[],long) method delegates method invocation to the stub's (*obj*) remote reference and allows the reference to take care of setting up the connection to the remote host, marshaling some representation for the *method* and parameters, *params*, then communicating the method invocation to the remote host.

This method either returns the result of the method invocation on the remote object which resides on the remote host or throws a RemoteException if the call failed or an application-level exception if the remote invocation throws an exception. Note that the operation number, *opnum*, represents a hash of the method signature and may be used to encode the method for transmission.

The method hash to be used for the *opnum* parameter is a 64-bit (long) integer computed from the first two 32-bit values of the message digest of a particular byte stream using the National Institute of Standards and Technology (NIST) Secure Hash Algorithm (SHA-1).

This byte stream contains a string as if it was written using the java.io.DataOutput.writeUTF method, consisting of the remote method's name followed by its method descriptor (see *The Java Virtual Machine Specification* (JVMS) for a description of method descriptors).

The ServerRef Interface

The interface ServerRef represents the server-side handle for a remote object implementation.

```
    package java.rmi.server;
```

```
public interface ServerRef extends RemoteRef {
        RemoteStub exportObject(java.rmi.Remote obj, Object data)
                throws java.rmi.RemoteException;
        String getClientHost() throws ServerNotActiveException;
}
```

The method exportObject finds or creates a client stub object for the supplied Remote object implementation *obj*.The parameter *data* contains information necessary to export the object (such as port number).

The method getClientHost returns the host name of the current client. When called from a thread actively handling a remote method invocation, the host name of the client invoking the call is returned. If a remote method call is not currently being service, then ServerNotActiveException is called.

The Skeleton Interface

The interface Skeleton is used solely by the implementation of skeletons generated by the rmic compiler. A skeleton for a remote object is a server-side entity that dispatches calls to the actual remote object implementation.

```
package java.rmi.server;

public interface Skeleton {
        void dispatch(Remote obj, RemoteCall call, int opnum, long hash)
                throws Exception;
        Operation[] getOperations();
}
```

The dispatch method unmarshals any arguments from the input stream obtained from the *call* object, invokes the method (indicated by the operation number *opnum*) on the actual remote object implementation *obj*, and marshals the return value or throws an exception if one occurs during the invocation. The get Operations method returns an array containing the operation descriptors for the remote object's methods.

The Operation Class

The class Operation holds a description of a method in the Java programming language for a remote object.

Note: The Operation interface is deprecated as of the Java 2 SDK, Standard Edition, v1.2. The 1.2 stub protocol no longer uses the old RemoteRef.invoke method which takes an

Operation as one of its arguments. As of the Java 2 SDK, Standard Edition, v1.2, stubs now use the new invoke method which does not require Operation as a parameter.

```
package java.rmi.server;

public class Operation {
	public Operation(String op) {...}
	public String getOperation() {...}
	public String toString() {...}
}
```

An Operation object is typically constructed with the method signature.

The method getOperation returns the contents of the operation descriptor (the value with which it was initialized).

The method toString also returns the string representation of the operation descriptor (typically the method signature).

DEFINING REMOTE OBJECTS

Now that you have a basic idea of how Java RMI works, we can explore the details of creating and using distributed objects with RMI. As I mentioned earlier, defining a remote RMI object involves specifying a remote interface for the object, then providing a class that implements this interface. The remote interface and implementation class are then used by RMI to generate a client stub and server skeleton for your remote object. The communication between local objects and remote objects is handled using these client stubs and server skeletons. The relationships among stubs, skeletons, and the objects that use them are shown

Relationships among remote object, stub, and skeleton classes

When a client gets a reference to a remote object (details on how this reference is obtained come later) and then calls methods on this object reference, there needs to be a way for the method request to get transmitted back to the actual object on the remote server and for the results of the method call to get transmitted back to the client. This is what the generated stub and skeleton classes are for. They act as the communication link between the client and your exported remote object, making it seem to the client that the object actually exists within its Java VM.

Remote objects are objects in a distributed computing environment that can be accessed and manipulated across different machines or network nodes.

These objects provide a way for software components to communicate and interact with each other over a network, regardless of their physical locations.

When defining remote objects, several aspects need to be considered:

Interface:

Remote objects define interfaces that specify the methods and attributes that can be accessed remotely. The interface acts as a contract between the client and the remote object, ensuring that both parties understand how to interact with each other.

Marshaling:

Remote objects utilize marshaling or serialization techniques to convert method calls, argument values, and return values into a format that can be transmitted over the network. This ensures that the objects can be properly transported across different machines and platforms.

Location transparency:

Remote objects enable transparent access to their methods and properties, hiding the complexities of the underlying network and communication protocols. This allows clients to invoke remote methods as if they were invoking local methods, simplifying the development and maintenance of distributed systems.

Remote object reference:

Remote objects are typically accessed through remote object references, which provide a means of identifying and locating the objects in a distributed system. These references are used by clients to establish a communication channel with the remote object and invoke its methods.

Activation and lifecycle management:

Remote objects may require activation and lifecycle management mechanisms to control their instantiation, deletion, and availability. These mechanisms ensure that remote objects are created, managed, and destroyed appropriately in a distributed environment.

Overall, defining remote objects involves designing interfaces, implementing the necessary communication protocols, and managing the lifecycle of these objects in a distributed system. This enables seamless interaction and collaboration between components located on different machines.

Example 3-3. The ThisOrThatServer Interface

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ThisOrThatServer extends Remote {
  public String doThis(String todo) throws RemoteException;
  public String doThat(String todo) throws RemoteException;
}
```

With the remote interface defined, the next thing we need to do is write a class that implements the interface. Example 3-4 shows the `ThisOrThatServerImpl` class, which implements the `ThisOrThatServer` interface.

Example 3-4. Implementation of the ThisOrThatServer

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class ThisOrThatServerImpl
  extends UnicastRemoteObject implements ThisOrThatServer {

  public ThisOrThatServerImpl() throws RemoteException {}

  // Remotely accessible methods
  public String doThis(String todo) throws RemoteException {
    return doSomething("this", todo);
  }

  public String doThat(String todo) throws RemoteException {
    return doSomething("that", todo);
  }

  // Non-remote methods
  private String doSomething(String what, String todo) {
    String result = "Did " + what + " to " + todo + ".";
    return result;
  }
```

This class has implementations of the do this () and do that () methods declared in this or That Server interface; it also has a no remote method, do something (), that is used to implement the two remote methods. Notice that the do something () method doesn't have to be declared as throwing a Remote Exception, since it isn't a remotely callable method. Only the methods declared in the remote interface can be invoked remotely. Any other methods you include in your implementation class are considered no remote (i.e., they are only callable from within the local Java virtual machine where the object exists).

Key RMI Classes for Remote Object Implementations

You probably noticed that our *ThisOrThatServerImpl* class also extends the *UnicastRemoteObject* class. This is a class in the *java.rmi.server* package that extends *java.rmi.server.RemoteServer*, which itself extends *java.rmi.ser-ver.RemoteObject,* the base class for all RMI remote objects. There are four key classes related to writing server object implementations:

Remote Object

RemoteObject implements both the Remote and *java.rmi.server* package, it is used by both the *Serializable* interfaces. Although the *RemoteObject* class is in the client and server portions of a remote object reference. Both client stubs and server implementations are *subclassed* (directly or indirectly) from *RemoteObject*. A *RemoteObject* contains the remote reference for a particular remote object.

*RemoteObjec*t is an abstract class that *reimplements* the equals (), hashCode(), and toString() methods inherited from Object in a way that makes sense and is practical for remote objects. The equals() method, for example, is implemented to return true if the internal remote references of the two *RemoteObject* objects are equal, (i.e., if they both point to the same server object).

Remote Server

RemoteServer is an abstract class that extends *RemoteObject*. It defines a set of static methods that are useful for implementing server objects in RMI, and it acts as a base class for classes that define various semantics for remote objects. In principle, a remote object can behave according to a simple point-to-point reference scheme; it can have replicated copies of itself scattered across the network that need to be kept synchronized; or any number of other scenarios. JDK 1.1 supported only point-to-point, *nonpersistent* remote references with the *UnicastRemoteObject* class. The Java 2 SDK 1.2 has introduced the RMI activation system, so it provides another subclass of *RemoteServer,* Activatable.

Unicast Remote Object

This is a concrete subclass of *RemoteServer* that implements point-to-point remote references over TCP/IP networks. These references are *nonpersistent*: remote references to a server object are only valid during the lifetime of the server object. Before the server object is created (inside a virtual machine running on the host) or after the object has been destroyed, a client can't obtain remote references to the object. In addition, if the virtual machine containing the object exits (intentionally or otherwise), any existing remote references on clients become invalid and generate *RemoteException* objects if used.

Activatable

This concrete subclass of *RemoteServer* is part of the new RMI object activation facility in Java 1.2 and can be found in the *java.rmi.activation* package. It implements a server object that supports persistent remote references. If a remote method request is received on the server host for an Activatable object, and the target object is not executing at the time, the object can be started automatically by the RMI activation daemon.

REMOTE OBJECT ACTIVATION

Prior to the release of the Java^{TM} 2 SDK, an instance
ofa UnicastRemoteObject could be accessed from a program that (1) created an instance of the remote object, and (2) ran all the time. Now with the introduction of the class java.rmi.activation.

Activatable and the RMI daemon, rmid, programs can be written to register information about remote object implementations that should be created and execute "on demand", rather than running all the time. The RMI daemon, rmid, provides a Java virtual machine* (JVM) from which other JVM instances may be spawned.

Starting out with activation

None of the activation tutorials should be the first material you read on RMI. If you have never used RMI before, you should take a look at *Getting Started* before you try these. The first three activation tutorials are intended for developers who have had some experience developing Java programs that access remote objects by means of the RMI classes provided as part of the Java 2 SDK.

Going further with activation

This next tutorial on using a Marshalled Object is intended for developers who have gone through at least one of the first three activation tutorials. More tutorials may be added to this section in a future release, based on your feedback.

- Using a MarshalledObject to create persistent data

Creating an Activatable Object

Prior to the release of the Java™ 2 SDK, an instance of a UnicastRemoteObject could be accessed from a server program that (1) created an instance of the remote object, and (2) ran all the time. Now with the introduction of the class java.rmi.activation.Activatable and the RMI daemon, rmid, programs can be written to register information about remote object implementations that should be created and execute "on demand," rather than running all the time. The RMI daemon, rmid, provides a Java virtual machine* (JVM) from which other JVM instances may be spawned.

The files needed for this tutorial are:

- Client.java, the class which will invoke a method on an activatable object
- MyRemoteInterface.java, the interface that extends java.rmi.Remote, implemented by:
- ActivatableImplementation.java, the class which will be activated
- Setup.java, the class which registers information about the activatable class with the RMI registry and the RMI daemon

You may notice that while the client code is included, it is not discussed in a step-by-step manner, like the implementation and setup classes. The reason for this omission, is that the client code for activatable objects is no different than the RMI client code for accessing non-activatable remote objects. Activation is strictly a server-side implementation decision.

Creating the implementation class

For this example, the implementation class will be examples.activation.ActivatableImplementation. There are four steps to create an implementation class:

1. Make the appropriate imports in the implementation class
2. Extend your class from java.rmi.activation.Activatable
3. Declare a two-argument constructor in the implementation class
4. Implement the remote interface method(s)

Step 1:
Make the appropriate imports in the implementation class

```
import java.rmi.*;
import java.rmi.activation.*;
```

Step 2:

Extend your class from java.rmi.activation.Activatable

```
public class ActivatableImplementation extends Activatable
    implements examples.activation.MyRemoteInterface {
```

Step 3:

Declare a two-argument constructor in the implementation class

```
public ActivatableImplementation(ActivationID id, MarshalledObject data)
    throws RemoteException {
    // Register the object with the activation system
    // then export it on an anonymous port
    super(id, 0);
}
```

Step 4:

Implement the remote interface method(s)

```
public Object callMeRemotely() throws RemoteException {
    return "Success";
}
```

Creating the "setup" class

The job of the "setup" class is to create all the information necessary for the activatable class, without necessarily creating an instance of the remote object. For this example the setup class will be examples.activation.Setup.

The setup class passes the information about the activatable class to rmid, registers a remote reference (an instance of the activatable class's stub class) and an identifier (name) with the rmiregistry, and then the setup class may exit. There are seven steps to create a setup class:

1. Make the appropriate imports
2. Install a SecurityManager
3. Create an ActivationGroup instance
4. Create an ActivationDesc instance
5. Declare an instance of your remote interface and register with rmid
6. Bind the stub to a name in the rmiregistry
7. Quit the setup application

Step 1:
Make the appropriate imports in the setup class

```
import java.rmi.*;
import java.rmi.activation.*
import java.util.Properties;
```

Step 2:
Install a SecurityManager

```
System.setSecurityManager(new RMISecurityManager());
```

Step 3:
Create an ActivationGroup instance

Note: In this example, for simplicity, we will use a policy file that gives global permission to anyone from anywhere. ***Do not use this policy file in a production environment***. For more information on how to properly open up permissions using a java.security.policy file, please refer to to the following documents:

Default Policy Implementation and Policy File Syntax

Permissions in the Java 2 SDK

In the setup application, the job of the activation group descriptor is to provide all the information that rmid will require to contact the appropriate existing JVM or spawn a new JVM for the activatable object.

Note: *In order to run this code on your system, you'll need to change the policy file location to be the absolute path to where you've installed the example policy file that came with the source code.*

```
// Because of the Java 2 security model, a security policy should
// be specified for the ActivationGroup VM. The first argument
// to the Properties put method, inherited from Hashtable, is
// the key and the second is the value
//
Properties props = new Properties();
props.put("java.security.policy",
   "/home/rmi_tutorial/activation/policy");

ActivationGroupDesc.CommandEnvironment ace = null;
ActivationGroupDesc exampleGroup = new ActivationGroupDesc(props, ace);
```

```
// Once the ActivationGroupDesc has been created, register it
// with the activation system to obtain its ID
//
ActivationGroupID agi =
   ActivationGroup.getSystem().registerGroup(exampleGroup);
```

Step 4:
Create an ActivationDesc instance

The job of the activation descriptor is to provide all the information that rmid will require to create a new instance of the implementation class.

Note: *In order to run this code on your system, you'll need to change the file URL location to be the location of the directory on your system, where you've installed the example source code.*

```
// The "location" String specifies a URL from where the class
// definition will come when this object is requested (activated).
// Don't forget the trailing slash at the end of the URL
// or your classes won't be found.
//
String location = "file:/home/rmi_tutorial/activation/";

// Create the rest of the parameters that will be passed to
// the ActivationDesc constructor
//
MarshalledObject data = null;

// The second argument to the ActivationDesc constructor will be used
// to uniquely identify this class; it's location is relative to the
// URL-formatted String, location.
//
ActivationDesc desc = new ActivationDesc
    ("examples.activation.ActivatableImplementation", location, data);
```

Step 5:
Declare an instance of your remote interface and register the activation descriptor with rmid

```
MyRemoteInterface mri = (MyRemoteInterface)Activatable.register(desc);
System.out.println("Got the stub for the ActivatableImplementation");
```

Step 6:
Bind the stub, that was returned by the Activatable.register method, to a name in the rmiregistry

```
Naming.rebind("ActivatableImplementation", mri);
System.out.println("Exported ActivatableImplementation");
```

Step 7:
Quit the setup application

```
System.exit(0);
```

Compile and run the code

There are six steps to compile and run the code:

1. Compile the remote interface, implementation, client, and setup classes
2. Run rmic on the implementation class
3. Start the rmiregistry
4. Start the activation daemon, rmid
5. Run the setup program
6. Run the client

Step 1:
Compile the remote interface, implementation, client and setup classes

```
% javac -d . MyRemoteInterface.java
% javac -d . ActivatableImplementation.java
% javac -d . Client.java
% javac -d . Setup.java
```

Step 2:

Run rmic on the implementation class

```
% rmic -d . examples.activation.ActivatableImplementation
```

Step 3:
Start the rmiregistry

```
% rmiregistry &
```

If you start the rmiregistry, *and it* can *find your stub classes in its CLASSPATH, it will ignore the server's* java.rmi.server.codebase *property, and as a result, your client(s) will not be able to download the stub code for your remote object.*

Step 4:
Start the activation daemon, rmid

% rmid -J-Djava.security.policy=rmid.policy &
Where rmid.policy is the name of the security policy file for rmid.

Note: By default, rmid now requires a security policy file, that is used to verify whether or not the information in each ActivationGroupDescriptor is allowed to be used to launch a JVM for an activation group. For complete details, please refer to the rmid man page for the Solaris operating environment and the rmid man page for the Microsoft Windows platform.

Step 5:
Run the setup program

Run the setup, setting the codebase property to be the location of the implementation stubs. There are four things that need to go on the same command line:

1. The "java" command
2. A property *name=value* pair that specifies the location of the security policy file
3. A property to specify where the stub code lives (no spaces from the "-D" all the way though the last "/")
4. The fully-qualified package name of the setup program.

There should be one space just after the word "java", one between the two properties, and a third one just before the word "examples" (which is very hard to see when you view this as text, in a browser, or on paper).

```
% java  -Djava.security.policy=/home/rmi_tutorial/activation/policy   -
Djava.rmi.server.codebase=file:/home/rmi_tutorial/activation/  examples.activation.Setup
```

The codebase property will be resolved to a URL, so it must have the form of "http://aHost/somesource/" or "file:/myDirectory/location/" or, due to the requirements of some operating systems, "file:///myDirectory/location/" (three slashes after the "file:").

While a file: URL is sometimes easier to use for running example code, using the file: URL will mean that the only clients that will be able to access the server are those that can access the same files system as the server (either by virtue of running on the same machine as the server or by using a shared filesystem, such as NFS)..

The server output should look like this:

```
Got the stub for the ActivatableImplementation
Exported ActivatableImplementation
```

Step 6:
Run the client

The argument to the examples.activation.Client program is the hostname of the implementation server, in this case, "vector".

```
% java -Djava.security.policy=/home/rmi_tutorial/activation/policy
 examples.activation.Client vector
```

The client output should look like this:

```
Got a remote reference to the object that extends Activatable.
Making remote call to the server
Returned from remote call
Result: Success
```

***As used on this web site, the terms "Java virtual machine" or "JVM" mean a virtual machine for the Java platform.**

SERIALIZATION AND DESERIALIZATION IN JAVA

Serialization in Java is a mechanism of *writing the state of an object into a byte-stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

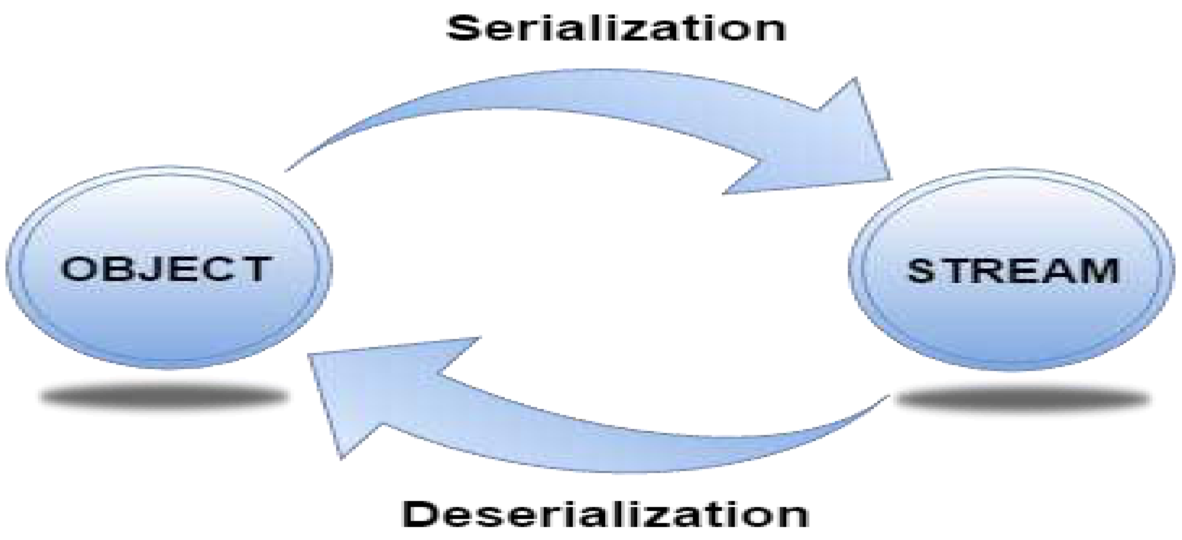
The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream* class, and for deserialization we call the **readObject()** method of *ObjectInputStream* class.

We must have to implement the *Serializable* interface for serializing the object.

Advantages of Java Serialization

It is mainly used to travel object's state on the network (that is known as marshalling).



java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The **Cloneable** and **Remote** are also marker interfaces.

The **Serializable** interface must be implemented by the class whose object needs to be persisted.

The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Let's see the example given below:

Student.java

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

In the above example, ***Student*** class implements Serializable interface. Now its objects can be converted into stream. The main class implementation of is showed in the next code.

Object Output Stream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

1) public ObjectOutputStream(OutputStream out) throws IOException {}	It creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	It creates an ObjectInputStream that reads from the specified InputStream.

Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	It reads an object from the input stream.
2) public void close() throws IOException {}	It closes ObjectInputStream.

Example of Java Serialization

In this example, we are going to serialize the object of ***Student*** class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

Persist.java

```java
import java.io.*;
class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
  out.close();
  System.out.println("success");
  }catch(Exception e){System.out.println(e);}
 }
}
```

Output:

```
success
```

THE JAVASPACES TECHNOLOGY

The Java Spaces technology is a high-level tool for building distributed applications, and it can also be used as a coordination tool. A marked departure from classic distributed models that rely on message passing or RMI, the JavaSpaces model views a distributed application as a collection of processes that cooperate through the flow of objects into and out of one or more spaces. This programming model has its roots in Linda, a coordination language developed by Dr. David Gelernter at Yale University. However, no knowledge of Linda is required to understand and use JavaSpaces technology.

The JavaSpaces service specification lists the following design goals for the JavaSpaces technology:

- It should provide a platform that simplifies the design and implementation of distributed computing systems.
- The client side should have few classes, both to keep the client simple and to speed the downloading of client classes.
- The client side should have a small footprint because it will run on computers with limited local memory.
- A variety of implementations should be possible.
- It should be possible to create a replicated JavaSpaces service.

JavaSpaces Technology vs. Databases

As mentioned earlier, a *space* is a shared network-accessible repository for objects: The data you can store there is persistent and later searchable. But a JavaSpaces service is not a relational or object database. JavaSpaces services are not used primarily as data repositories. They are designed for a different purpose than either relational or object databases.
Although a JavaSpaces service functions somewhat like a file system and somewhat like a database, it is neither. The key differences between JavaSpaces technology and databases are the following:

- ***Relational databases*** - understand the data they store and manipulate it directly through query languages such as SQL. JavaSpaces services, on the other hand, store entries that they understand only by type and the serialized form of each field. As a result, there are no general queries in the JavaSpaces application design, only "exact match" or "don't care" for a given field.
- ***Object databases*** - provide an object-oriented image of stored data that can be modified and used, almost as if it were transient memory. JavaSpaces systems do not provide a nearly transparent persistent or transient layer, and they work only on copies of entries.

Java Spaces Services and Operations

Application components (or processes) use the persistent storage of a space to store objects and to communicate. The components coordinate actions by exchanging objects through spaces; the objects do not communicate directly. Processes interact with a space through a simple set of operations.

You can invoke four primary operations on a Java Spaces service:

- write(): Writes new objects into a space
- take(): Retrieves objects from a space
- read(): Makes a copy of objects in a space

- notify: Notifies a specified object when entries that match the given template are written into a space

Both the read() and take() methods have variants: readIfExists() and takeIfExists(). If they are called with a zero timeout, then they are equivalent to their counterpart. The timeout parameter comes into effect only when a transaction is used.

Each operation has parameters that are entries. Some are templates, which are a kind of entry. The write() operation is a store operation. The read() and take() operations are a combination of search and fetch operations. The notify method sets up repeated search operations as entries are written to the space. If a take() or read() operation doesn't find an object, the process can wait until an object arrives.

Unlike conventional object stores, objects are passive data. Therefore, processes do not modify objects in the space or invoke their methods directly. In order to modify an object, a process must explicitly remove, update, and reinsert it into the space.

The space itself provides a set of key features.

The JavaSpaces Technology Application Model

A JavaSpaces service holds entries, each of which is a typed group of objects expressed in a class that implements the interface net.jini.core.entry.

Entry. Once an entry is written into a JavaSpaces service, it can be used in future look-up operations. Looking up entries is performed using templates, which are entry objects that have some or all of their fields set to specified values that must be matched exactly.

All remaining fields, which are not used in the lookup, are left as wildcards.

There are two look-up operations: read() and take(). The read() method returns either an entry that matches the template or an indication that no match was found.

The take() method operates like read(), but if a match is found, the entry is removed from the space. Distributed events can be used by requesting a Java Spaces service to notify you when an entry that matches the specified template is written into the space.

Note that each entry in the space can be taken at most once, but two or more entries may have the exact same values.

Using JavaSpaces technology, distributed applications are modeled as a flow of objects between participants, which is different from classic distributed models such as RMIs. indicates what a JavaSpaces technology-based application looks like.

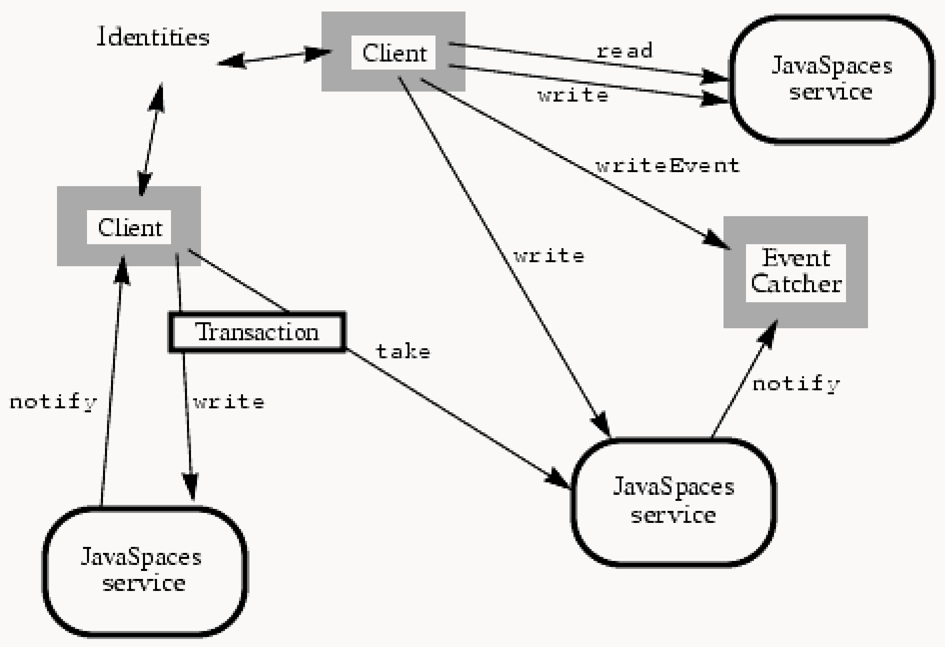


Figure 1: A Typical JavaSpaces Technology Application

As you can see, a client can interact with as many JavaSpaces services as needed. Clients perform operations that map entries to templates onto JavaSpaces services. Such operations can be singleton or contained in a transaction so that all or none of the operations take place. Notifications go to event catches, which can be either clients or proxies for clients.

To get a flavor of how to implement distributed applications using a handful of JavaSpaces operations, consider a multiuser chat system. All the messages that make up the discussion are written to a space that acts as a chat area. Participants write message objects into the space, while other members wait for new message objects to appear, then read them out and display their contents. The list of participants can be kept in the space and updated whenever someone joins or leaves the discussion. Because the space is persistent, a new member can read and view the entire discussion.

You can implement such a multiuser chat system in RMI by creating remote interfaces for the interactions discussed. But by using JavaSpaces technology, you need only one interface.

The JavaSpaces Technology Programming Model

All operations are invoked on an object that implements the net.jini.space.JavaSpace interface. A space stores entries, each of which is a collection of typed objects that implements the Entry interface. Code Sample 1 shows a MessageEntry that contains one field: content, which is null by default. Information on how to compile and run the sample application appears later in this article.

Code Sample 1: MessageEntry.java

```
import net.jini.core.entry.*;

  public class MessageEntry implements Entry {
    public String content;

    public MessageEntry() {
    }

    public MessageEntry(String content) {
      this.content = content;
    }

    public String toString() {
      return "MessageContent: " + content;
    }
  }
```

MessageEntryspace

```
JavaSpace space = getSpace();
   MessageEntry msg = new MessageEntry();
   msg.content = "Hello there";
   space.write(msg, null, Lease.FOREVER);
```

nullTransaction

The write() operation places a copy of an entry into the given JavaSpace service, and the Entry passed is not affected by the operation. Each write() operation places a new Entry into the space even if the same Entry object is used in more than one write().

Entries written in a space are governed by a renewable lease. If you like, you can change the lease (when the write() operation is invoked) to one hour as follows: > space.write(msg, null, 60 * 60 * 1000);

write()Lease

Once the entry exists in the space, any process with access to the space can perform a read() on it. To read an entry, a template is used, which is an entry that may have one or more of its fields set to null. An entry matches a template if (a) the entry has the same type as or is a subtype of the template and (b) if for every specified non- null field in the template, their fields match exactly. The null fields act as wildcards and match any value. The following code segment shows how to create a template and perform a read() on the space:

```
MessageEntry template = new MessageEntry();
         MessageEntry output = (MessageEntry) space.read(template, null, 
Long.MAX_VALUE);
```

null MessageEntry Long.MAX_VALUE read() take() readIfExists()

Code Sample 2 shows the client that discovers the JavaSpace service, writes a message into the space, and then reads it. Instructions on how to compile and run this sample application appear later in this article.

Code Sample 2: SpaceClient.java

```
import net.jini.space.JavaSpace;

           public class SpaceClient {
             public static void main(String argv[]) {
               try {
                 MessageEntry msg = new MessageEntry();
                 msg.content = "Hello there";
                 System.out.println("Searching for a JavaSpace...");

                        Lookup finder = new Lookup(JavaSpace.class);
                 JavaSpace space = (JavaSpace) finder.getService();
                 System.out.println("A JavaSpace has been discovered.");
                 System.out.println("Writing a message into the space...");
                 space.write(msg, null, 60*60*1000);
                 MessageEntry template = new MessageEntry();
                 System.out.println("Reading a message from the space...");
                 MessageEntry result = (MessageEntry) space.read(template, null,
Long.MAX_VALUE);
                 System.out.println("The message read is: "+result.content);
               } catch(Exception e) {
                 e.printStackTrace();
               }
             }
           }
```

Transactions

The JavaSpaces API uses the package net.jini.core.transaction to provide basic atomic transactions that group multiple operations across multiple JavaSpaces services into a bundle that acts as a single atomic operation. Either all modifications within the transactions will be applied or none will, regardless of whether the transaction spans one or more operations or one or more JavaSpaces services. Note that transactions can span multiple spaces and participants in general.

A read(), write(), or take() operation that has a null transaction acts as if it were in a committed transaction that contained that operation. As an example, a take() with a null transaction parameter performs as if a transaction was created, the take() was performed under that transaction, and then the transaction was committed.

The Jini Outrigger JavaSpaces Service

The Jini Technology Starter Kit comes with the package com.sun.jini.outrigger, which provides an implementation of a JavaSpaces technology-enabled service. You can run it two ways:

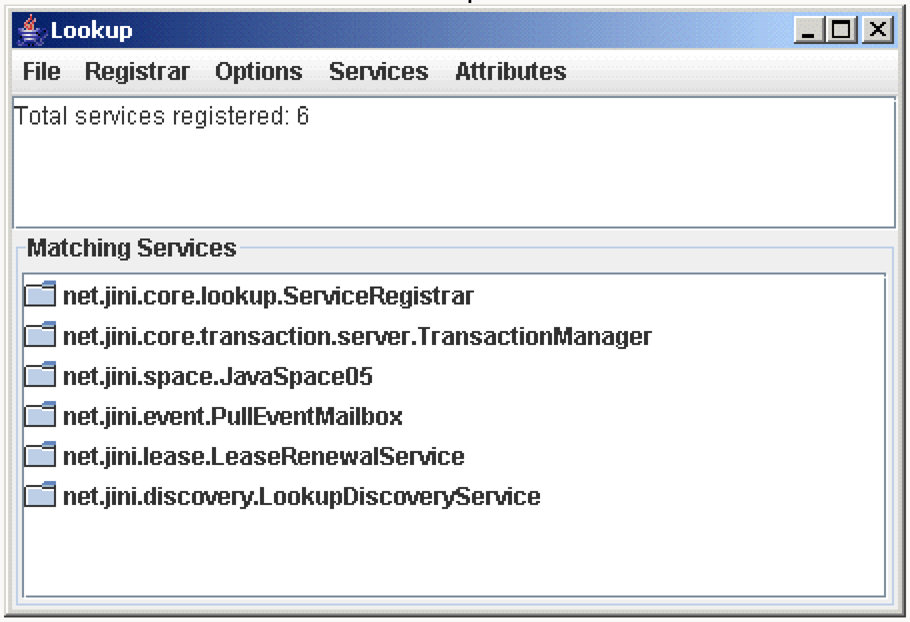
- As a transient space that loses its state between executions:
  Use com.sun.jini.outrigger.TransientOutriggerImpl.
- As a persistent space that maintains state between executions:
  Use com.sun.jini.outrigger.PersistentOutriggerImpl.

TransientOutriggerImplPersistentOutriggerImpl

Compiling and Running the SpaceClient Application

To compile and run the sample application in this article, do the following:

1. Compile the code in Code Sample 1 (MessageEntry.java) using javac as follows:
   prompt> javac -classpath <pathToJiniInstallation\lib\jini-ext.jar> MessageEntry.java
   Note that you need to include the JAR file jini-ext.jar in your classpath. This JAR file comes with the starter kit and is in the lib directory of your installation.
2. Compile the code in Code Sample 2 (SpaceClient.java). Note that this code makes use of a utility class called Lookup to locate or discover a JavaSpace space. Therefore, before you compile SpaceClient.java, you should download Lookup.java and then compile both classes using javac as shown in step 2. Note that you should include the directory that contains MessageEntry.class in your classpath when compiling SpaceClient.java.
3. Run Launch-All, which is in the installverify directory of your Jini installation directory. This will start a service browser (as shown in Figure 2) and six contributed Jini network technology services, one of which is the JavaSpace service.



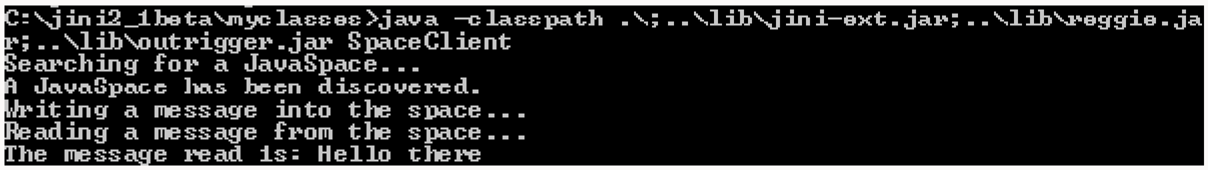
```
Total services registered: 6
```

```
net.jini.core.lookup.ServiceRegistrar
net.jini.core.transaction.server.TransactionManager
net.jini.space.JavaSpace05
net.jini.event.PullEventMailbox
net.jini.lease.LeaseRenewalService
net.jini.discovery.LookupDiscoveryService
```

Figure 2: Jini Network Technology Service Browser

4. Finally, run the SpaceClient application using the java command as follows. Here I assume that your Jini installation directory is C:\Jini2_1beta and that the classes you compiled earlier are at C:\Jini2_1beta\myclasses.
5. C:\Jini2_1beta\myclasses> java -classpath .\;
   ..\lib\jini-ext.jar;..\lib\reggie.jar;..\lib\outrigger.jar SpaceClient

   If all goes well, you will see the output shown in Figure 3.



```
C:\jini2_1beta\myclasses>java -classpath .\;..\lib\jini-ext.jar;..\lib\reggie.ja
r;..\lib\outrigger.jar SpaceClient
Searching for a JavaSpace...
A JavaSpace has been discovered.
Writing a message into the space...
Reading a message from the space...
The message read is: Hello there
```

Figure 3: SpaceClient Sample Output

Conclusion

The JavaSpaces technology provides services and tools for building sophisticated distributed applications. This technology is designed to work with applications that can model themselves as flow objects through one or more servers. If your application can be modeled this way, JavaSpaces technology will provide you with many benefits, such as a reliable distributed storage system for the objects. In addition, JavaSpaces technology handles concurrent access, storing and retrieving entries atomically.

UNIT-3

DATABASE

JAVA IN DATABASES:

Java is not a database itself, but it can be used to interact with databases. Java provides various libraries and frameworks for accessing and manipulating databases. Some popular Java libraries for database interaction include:

JDBC (Java Database Connectivity):
It is the standard API for connecting and working with relational databases. JDBC allows Java applications to execute SQL queries, retrieve and update data, and perform various database operations.

Hibernate:
It is an object-relational mapping (ORM) framework for Java. Hibernate simplifies the process of mapping Java objects to relational database tables and provides an abstraction layer for database operations.

JPA (Java Persistence API):
JPA is a Java specification for ORM. It provides a higher-level abstraction for managing database persistence.

Apache Cassandra:
It is a highly scalable NoSQL database. Java drivers are available to interact with Cassandra and perform operations like inserting, updating, and querying data.

MongoDB:
MongoDB is a popular document-based NoSQL database. MongoDB provides a Java driver that allows Java applications to interact with MongoDB and perform CRUD operations.

These are just a few examples, and there are many more databases and corresponding Java libraries available. The choice of library or framework depends on the specific database and requirements of the application.

Java is commonly used in databases for a variety of reasons. Some of the key uses of Java in databases include:

JDBC (Java Database Connectivity): Java provides a robust API called JDBC, which allows developers to connect to various databases, execute SQL queries, and retrieve or update data. JDBC offers a standardized way to interact with databases using Java.

Object-Relational Mapping (ORM) frameworks: Java has numerous ORM frameworks such as Hibernate, MyBatis, and EclipseLink, which facilitate the mapping of Java objects to database tables. These frameworks simplify database operations by eliminating the need for writing complex SQL queries.

Stored procedures: Java can be used to write and execute stored procedures in databases. Stored procedures are pre-compiled database queries that are stored on the database server. They improve performance and security by reducing network traffic and allowing developers to encapsulate business logic within the database.
Data access layer: Java can be used to create a data access layer, which serves as an abstraction between the application and the database. This layer handles all the interactions with the database, allowing developers to focus on business logic without worrying about low-level database operations.

Database interfaces: Java can provide a user-friendly interface for managing and interacting with databases. Web-based applications built with Java can have forms, tables, and reports for performing CRUD (Create, Read, Update, and Delete) operations on the database.

Overall, Java's strength lies in its versatility and ability to integrate with various databases and database technologies, making it a popular choice for developing database-driven applications.

Advantages of Java in the database

Adaptive Server provides a runtime environment for Java, which means that Java code can be executed in the server. Building a runtime environment for Java in the database server provides powerful new ways of managing and storing both data and logic.

- You can use the Java programming language as an integral part of Transact-SQL.
- You can reuse Java code in the different layers of your application—client, middle-tier, or server—and use them wherever makes most sense to you.
- Java in Adaptive Server provides a more powerful language than stored procedures for building logic into the database.
- Java classes become rich, user-defined data types.
- Methods of Java classes provide new functions accessible from SQL.
- Java can be used in the database without jeopardizing the integrity, security, and robustness of the database. Using Java does not alter the behavior of existing SQL statements or other aspects of non-Java relational database behavior.

JDBC PRINCIPLES:

JDBC (Java Database Connectivity) is a Java API that allows Java applications to interact with databases. It follows certain principles:

Driver Manager: The JDBC Driver Manager is responsible for managing the drivers for the different databases. It loads and unloads drivers dynamically at runtime.

Connection: The Connection interface represents the logical connection between a Java application and a database. It provides methods for establishing, closing, and managing the connection.

Statement: The Statement interface is used to execute SQL queries or updates against a database. It can be used for both static and dynamic SQL statements.

Prepared Statement: The Prepared Statement interface is a sub interface of Statement that allows recompilation of SQL queries. It provides improved performance and security by separating the SQL statement from the data.

Result Set: The Result Set interface represents the result of a database query. It provides methods for accessing and manipulating the data retrieved from the database.

Transaction Management: JDBC supports transaction management using the Connection interface. It provides methods to start, commit, and rollback transactions.

Exception Handling: JDBC uses checked exceptions for database-related errors, such as SQLSyntaxErrorException and SQLException. Applications must handle these exceptions using try-catch blocks or propagate them to the caller.

Connection Pooling: JDBC supports connection pooling, which allows multiple applications to share a pool of database connections. This improves performance and resource utilization.

Metadata: JDBC provides metadata interfaces, such as Database Meta Data and Result Set Meta Data, for retrieving information about the database structure and query results.

Batch Processing: JDBC supports batch processing, which allows multiple SQL statements to be submitted as a single unit. This improves performance by reducing the number of round trips to the database.

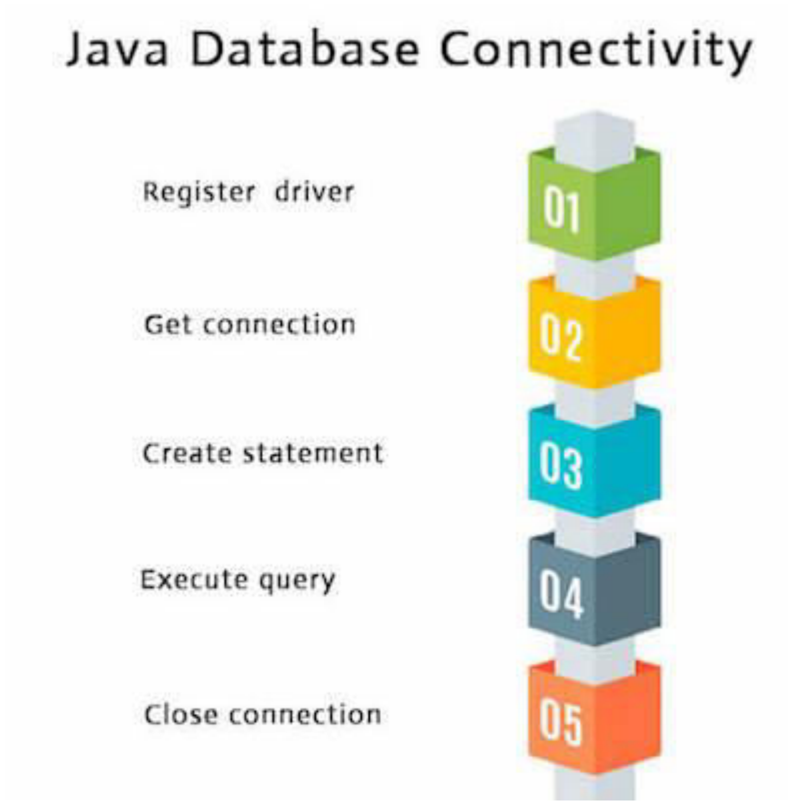
Overall, JDBC follows the principles of encapsulation, modularity, and flexibility, making it a powerful and widely used API for database connectivity in Java applications.

Java Database Connectivity with 5 Steps

1. 5 Steps to connect to the database in java
1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

> ***Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.***

Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

```
1. Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

```
1.    1) public static Connection getConnection(String url)throws SQLException
2.    2) public static Connection getConnection(String url,String name,String pas
sword)
3.    throws SQLException
```

Example to establish connection with the Oracle database

```
1.    Connection con=DriverManager.getConnection(
2.    "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
1.    public Statement createStatement()throws SQLException
```

Example to create the statement object

```
1.    Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```java
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```java
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```java
public void close()throws SQLException
```

Example to close connection

```java
con.close();
```

Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.

It avoids explicit connection closing step.

JDBC

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

Fundamentally, JDBC is a specification that provides a complete set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executables, such as −

- Java Applications
- Java Applets
- Java Servlets
- Java ServerPages (JSPs)
- Enterprise JavaBeans (EJBs).

All of these different executables are able to use a JDBC driver to access a database, and take advantage of the stored data.

JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code.

Pre-Requisite

Before moving further, you need to have a good understanding of the following two subjects −

- Core JAVA Programming
- SQL or MySQL Database

JDBC Architecture

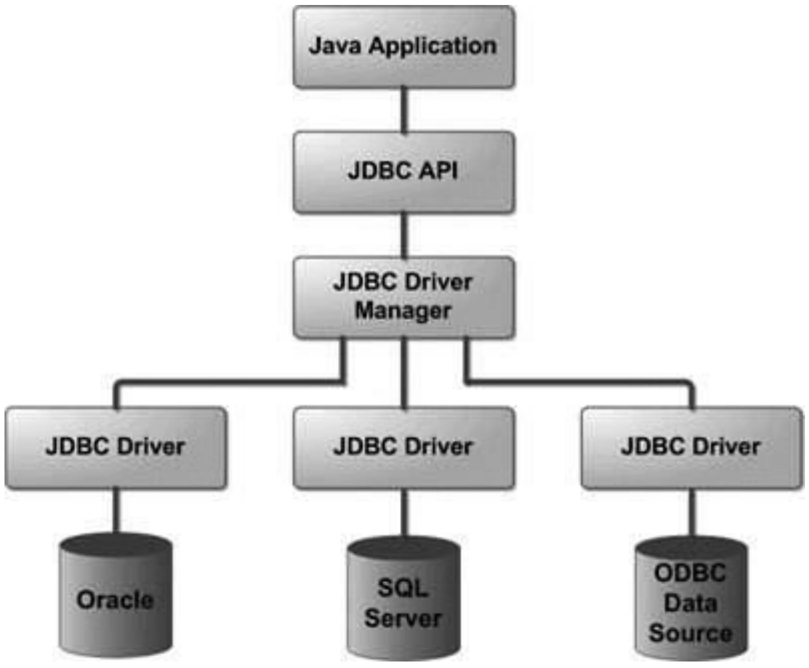
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API** − This provides the application-to-JDBC Manager connection.
- **JDBC Driver API** − This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **Driver Manager** − this class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.

- **Driver** − this interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use Driver Manager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection** − this interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement** − you use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **Result Set** − these objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

-

- **SQL Exception** − this class handles any errors that occur in a database application.

The JDBC 4.0 Packages

The java.sql and javax.sql are the primary packages for JDBC 4.0. This is the latest JDBC version at the time of writing this tutorial. It offers the main classes for interacting with your data sources.

The new features in these packages include changes in the following areas −

- Automatic database driver loading.
- Exception handling improvements.
- Enhanced BLOB/CLOB functionality.
- Connection and statement interface enhancements.
- National character set support.
- SQL ROWID access.
- SQL 2003 XML data type support.
- Annotations.
-

Database access

Java™ programs can access database files in several ways.

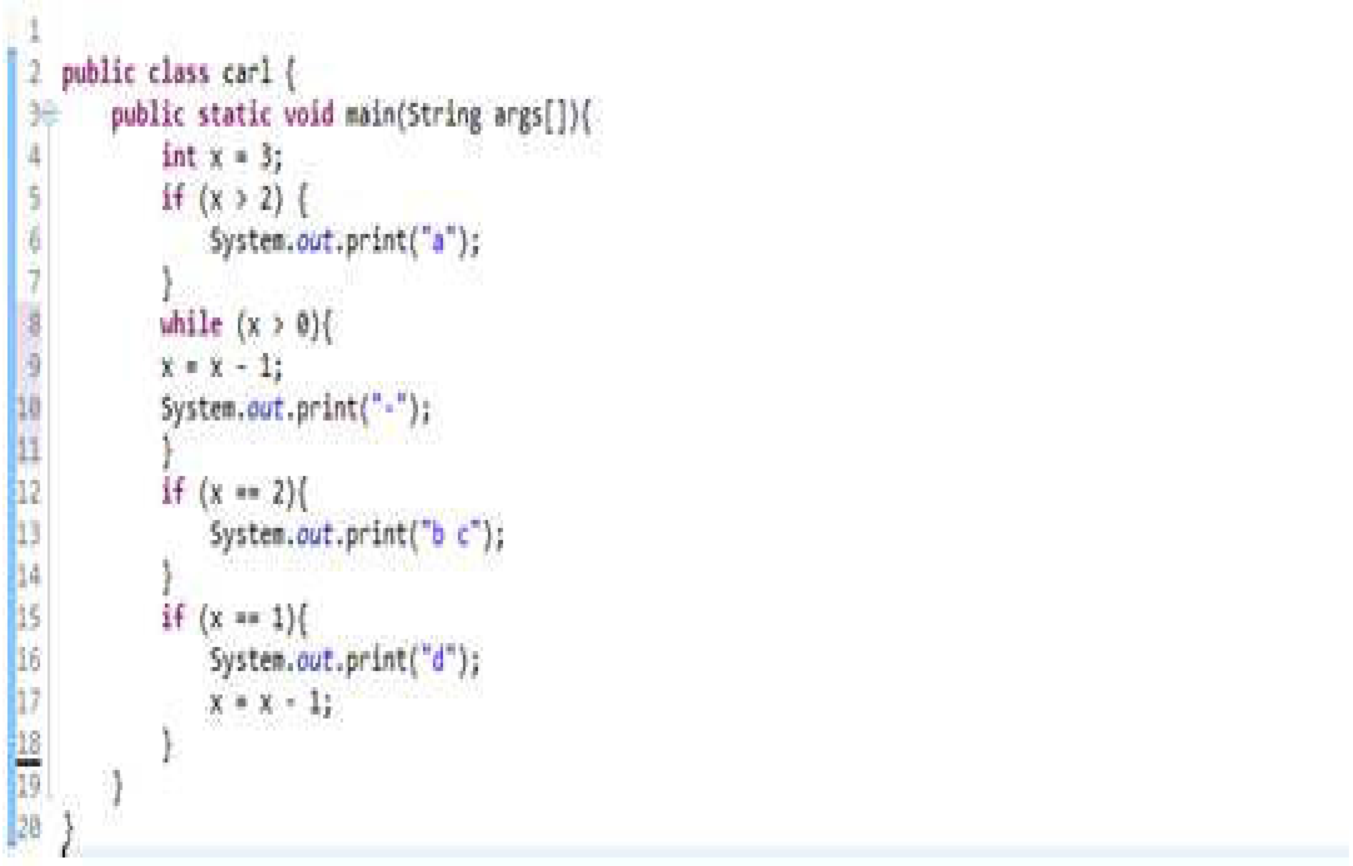
- **Accessing your IBM i database with the Java JDBC driver**
  The Java JDBC driver, also known as the "native" driver, provides programmatic access to IBM i database files. Using the Java Database Connectivity (JDBC) API, applications written in the Java language can access JDBC database functions with embedded Structured Query Language (SQL), run SQL statements, retrieve results, and propagate changes back to the database. The JDBC API can also be used to interact with multiple data sources in a distributed, heterogeneous environment.
- **Accessing databases using DB2 SQLJ support**
  DB2® Structured Query Language for Java (SQLJ) support is based on the SQLJ ANSI standard. The DB2 SQLJ support is contained in the IBM Developer Kit for Java. DB2 SQLJ support allows you to create, build, and run embedded SQL for Java applications.
- **Java SQL routines**
  your system provides the ability to access Java programs from SQL statements and programs. This can be done using Java stored procedures and Java user-defined functions (UDFs). The IBM i supports both the DB2 and SQLJ conventions for calling Java stored procedures and Java UDFs. Both Java stored procedures and Java UDFs can use Java classes that are stored in JAR files. The IBM i uses stored procedures defined by the *SQLJ Part 1* standard to register JAR files with the database.

INTERACTING:
JDBC API is a Java API which is used to interact with traditional databases like RDBMS. This JDBC works with wide variety of platforms like Windows, Mac etc. This JDBC supports two-tier and three-tier architecture for processing the data. It implements the JDBC interfaces for interacting with servers like Databases.

Driver :
A driver is nothing but an engine, which especially drives for the data.

For making an interaction, we usually have 4 types of drivers for connecting the java with database objects. Let me explain you one – by –one in detail in how do java and SQL interact with each other

Type – 1 :
This driver is also called a JDBC –ODBC bridge driver .using this driver, we can access the ODBC drivers installed on each machine. ODBC requires the configuration of a system a Data Source name, that represents the target database
The architecture of this driver is shown below



```

public class carl {
    public static void main(String args[]){
        int x = 3;
        if (x > 2) {
            System.out.print("a");
        }
        while (x > 0){
        x = x - 1;
        System.out.print("-");
        }
        if (x == 2){
            System.out.print("b c");
        }
        if (x == 1){
            System.out.print("d");
            x = x - 1;
        }
    }
}
```

Type – 2 :

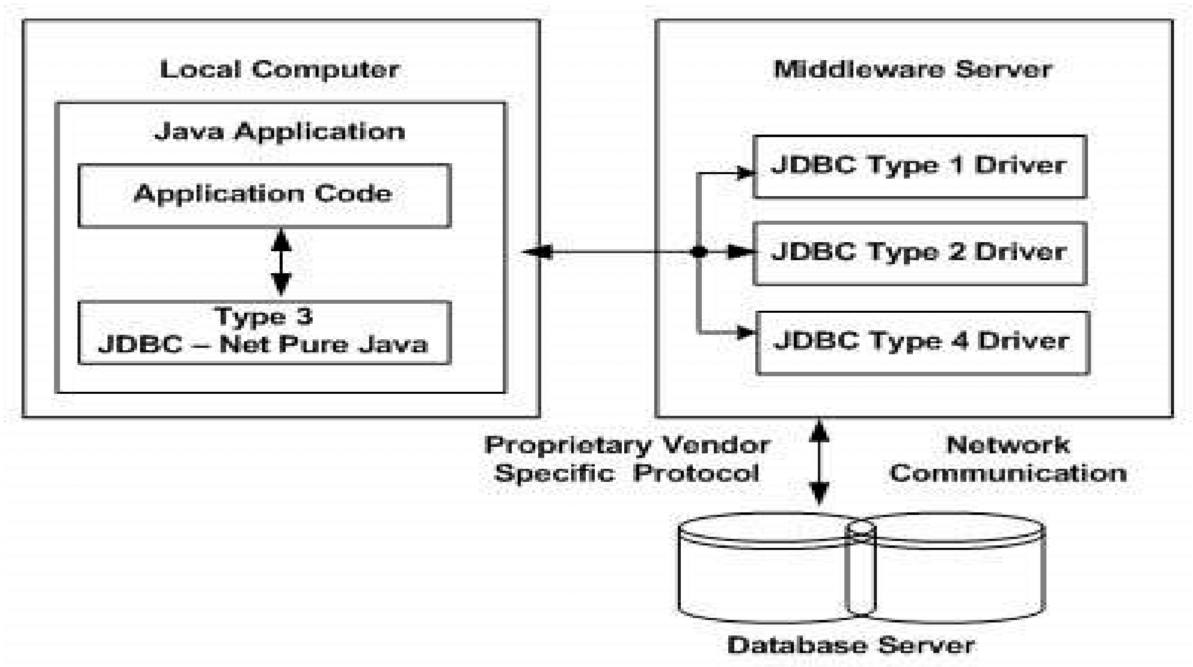
This type of driver is called the Native API .here the JDBC API calls were converted into Native API C /C ++ API calls, which were unique in the database.

These drivers were being provided by different database vendors. This driver works just like Type -1 driver. But the vendor specific driver must be installed on each client machine. Moreover, one thing we need to remember is that we need to change the native API for every change in database.

Type – 3 :

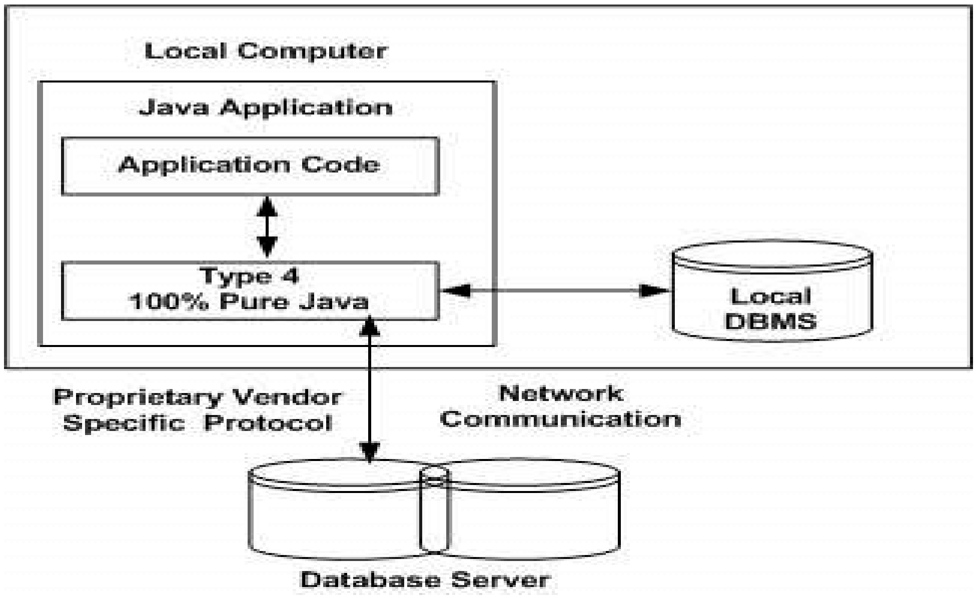
This type of driver is called JDBC-.Net pure Java. Here a three-tier approach is used for accessing the databases. This JDBC uses the standard client sockets for transferring the data into a middleware.

Then, this data will be accessed by the end user. The greatest advantage of this type of driver, no code is required for accessing the data. Moreover, a single driver can give access to multiple databases



Type – 4 :

This type of driver is known as 100% pure JAVA. It is used for accessing the vendor's databases directly through a socket. This s the highest performance driver available for databases and is usually provided by a vendor itself.



My dear readers, I hope you got a clarity regarding the types of drivers sued in JDBC connection. Moreover, I hope you got an idea regarding how do JAVA and SQL interact with each other.

Recommended Audience:

- Software developers
- Project Managers
- Team leaders

Pre requisites:

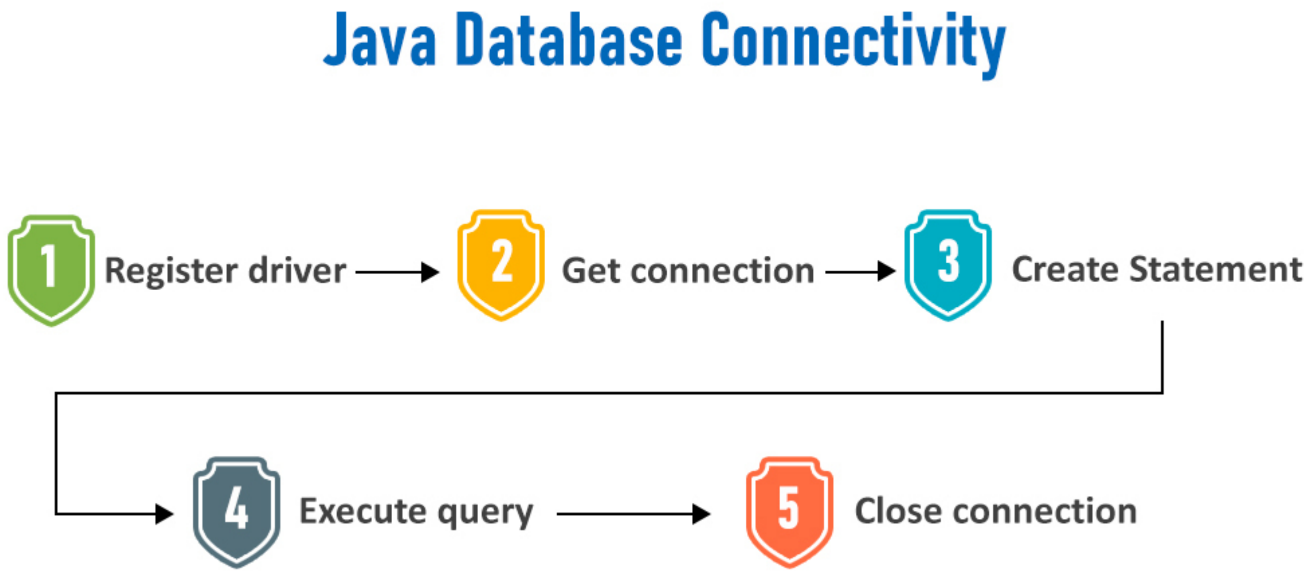
There are nothing much prerequisites requires in order to pursue the **JAVA COURSE**. It good to have a basic knowledge of C / C++. But not mandatory. Trainers of Online IT Guru will teach you from the basics if you don’t have a knowledge of these concerts.

Java Program to Search the Contents of a Table in JDBC

- **Read**
- Discuss
- Courses
- Practice

In order to deal with JDBC standard 7 steps are supposed to be followed:

1. Import the database
2. Load and register drivers
3. Create a connection
4. Create a statement
5. Execute the query
6. Process the results
7. Close the connection



Procedure:

1. **Import the database**-syntax for importing the sql database in java is-
   import java.sql.* ;

2. **Load and register drivers**-syntax for registering drivers after the loading of driver class is
   forName(com.mysql.jdbc.xyz) ;

3. **Creating a database** irrespective of *SQL* or *NoSQL*. Creating a database using *sqlyog* and creating some tables in it and fill data inside it in order to search for the contents of a table. For example, the database is named as "hotelman" and table names are "cuslogin" and "adminlogin".

4. **Create a connection:** Open any IDE where the java executable file can be generated following the standard methods. Creating a package further creating the class. Inside the package, open a new java file and type the below code for JDBC connectivity and save the filename with connection.java.

5. **Searching content** in a table, let's suppose my "cuslogin" table has columns namely "id", "name", "email", "password" and we want to search the customer whose id is 1.

6. Initialize a string with the SQL query as follows

String sql="select * from cuslogin where id=1";

If we want to search for any id in general, then the SQL query becomes

String sql="select * from cuslogin where id="+Integer.parseInt(textfield.getText());

The textfield is the area(in Jframe form) where the user types the id he wants to search in the "cuslogin" table.

4.1: Initialize the below objects of Connection class, Prepared Statement class, and Result Set class(needed for JDBC) and connect with the database as follows

Connection con = null;

PreparedStatement p = null;

ResultSet rs = null;

con = connection.connectDB();

4.2: Now, add the SQL query of step 3.1 inside prepareStatement and execute it as follows:

p =con.prepareStatement(sql);

rs =p.executeQuery();

4.3: We check if rs.next() is not null, then we display the details of that particular customer present in "cuslogin" table

4.4: Open a new java file (here, its result.java) inside the same package and type the full code (shown below) for searching the details of the customer whose id is 1, from table "cuslogin".

Note: *both the file viz result.java and connection.java should be inside the same package, else the program won't give desired output!!*

Implementation:
Example 1
Connection class of JDBC by making an object to be invoked in main(App) java program below in 1B

- Java

```
// Java program to search the contents of

// a table in JDBC Connection class for JDBC

// Connection class of JDBC



// Importing required classes

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;



public class connectionDB {



    final String DB_URL

        = "jdbc:mysql://localhost:3306/testDB?useSSL=false";



    //  Database credentials
```

```
 // We need two parameters to access the database

 // Root and password



 // 1. Root

 final String USER = "root";



 // 2. Password to fetch database

 final String PASS = "Imei@123";



 // Connection class for our database connectivity

public Connection connectDB()

 {

     // Initially setting NULL

     // to connection class object

     Connection con = null;



     // Try block to check exceptions

     try {
```

```
    // Loading DB(SQL) drivers

    Class.forName("com.mysql.cj.jdbc.Driver");



    // Registering SQL drivers

    con = DriverManager.getConnection(DB_URL, USER,

                                PASS);

}



// Catch block to handle database exceptions

catch (SQLException e) {



    // Print the line number where exception occurs

    e.printStackTrace();

}



// Catch block to handle exception

// if class not found

catch (ClassNotFoundException e) {
```

```
            // Function prints the line number
            // where exception occurs
            e.printStackTrace();
        }

        // Returning Connection class object to
        // be used in (App/Main) GFG class
        return con;
    }
}
```

App/Main Class where the program is compiled and run calling the above connection class object

- Java

```
// Java program to Search the
// contents of a table in JDBC

// Main Java program (App Class) of JDBC
// Step 1: Importing database files

// Importing SQL libraries

import java.sql.*;



// Main class

// It's connection class is shown above

public class GFG {



    // Main driver method

    public static void main(String[] args)

    {

        // Step 2: Establishing a connection

        connectionDB connection = new connectionDB();



        // Assigning NULL to object of Connection class

        // as shown returned by above program

        Connection con = null;

        PreparedStatement p = null;
```

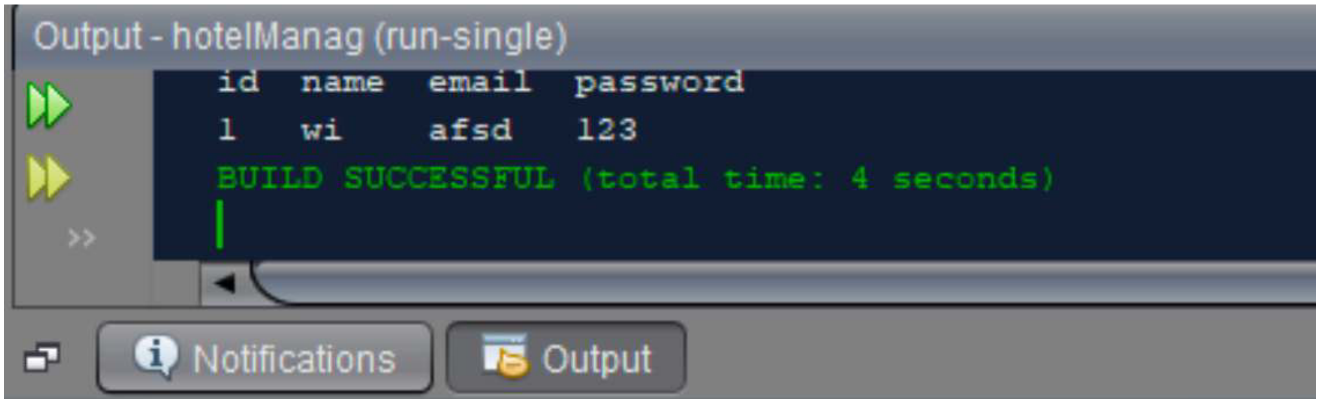
```
ResultSet rs = null;



// Step 3: Loading and registereding drivers

// Loaded and registered in Connection class

// shown in above program

con = connection.connectDB();



// Try block to check exceptions

try {



    // Step 4: Write a statement

    String sql

        = "select * from cuslogin where id=1";



    // Step 5: Execute the query

    p = con.prepareStatement(sql);

    rs = p.executeQuery();



    // Step 6: Process the results
```

}

```
    System.out.println(
        "id\t\tname\t\temail\t\tpassword");


    // Condition check using next() method
    // Holds true till there is single element remaining
   // in the object
    if (rs.next()) {


        int id = rs.getInt("id");
        String name = rs.getString("name");
        String email = rs.getString("email");
        String password = rs.getString("password");


      // Print and display name, emailID and password
      System.out.println(id + "\t\t" + name
                          + "\t\t" + email + "\t\t"
                          + password);
    }
  }
```

```
        // Catch block to handle exceptions

        catch (SQLException e) {



            // Print the exception

            System.out.println(e);

        }

    }

}
```

Output: Based on the values stored inside the "cuslogin" table.



```
id   name   email   password
1    wi     afsd    123
BUILD SUCCESSFUL (total time: 4 seconds)
```

MULTIMEDIA DATABASE

Multimedia database is the collection of interrelated multimedia data that includes text, graphics (sketches, drawings), images, animations, video, audio etc and have vast amounts of multisource multimedia data. The framework that manages different types of multimedia data which can be stored, delivered and utilized in different ways is known as multimedia database management system. There are three classes of the multimedia database which includes static media, dynamic media and dimensional media.

Content of Multimedia Database management system:

1. **Media data –** The actual data representing an object.

2. **Media format data –** Information such as sampling rate, resolution, encoding scheme etc. about the format of the media data after it goes through the acquisition, processing and encoding phase.

3. **Media keyword data –** Keywords description relating to the generation of data. It is also known as content descriptive data. Example: date, time and place of recording.

4. **Media feature data –** Content dependent data such as the distribution of colors, kinds of texture and different shapes present in data.

Types of multimedia applications based on data management characteristic are:

1. **Repository applications –** A Large amount of multimedia data as well as meta-data (Media format date, Media keyword data, and Media feature data) that is stored for retrieval purpose, e.g., Repository of satellite images, engineering drawings, and radiology scanned pictures.

2. **Presentation applications –** They involve delivery of multimedia data subject to temporal constraint. Optimal viewing or listening requires DBMS to deliver data at certain rate offering the quality of service above a certain threshold. Here data is processed as it is delivered. Example: Annotating of video and audio data, real-time editing analysis.

3. **Collaborative work using multimedia information –** It involves executing a complex task by merging drawings, changing notifications. Example: Intelligent healthcare network.

There are still many challenges to multimedia databases, some of which are :

1. **Modelling –** Working in this area can improve database versus information retrieval techniques thus, documents constitute a specialized area and deserve special consideration.

2. **Design –** The conceptual, logical and physical design of multimedia databases has not yet been addressed fully as performance and tuning issues at each level are far more complex as they consist of a variety of formats like JPEG, GIF, PNG, MPEG which is not easy to convert from one form to another.

3. **Storage –** Storage of multimedia database on any standard disk presents the problem of representation, compression, mapping to device hierarchies, archiving and buffering during input-output operation. In DBMS, a"BLOB" (Binary Large Object) facility allows untyped bitmaps to be stored and retrieved.

4. **Performance –** For an application involving video playback or audio-video synchronization, physical limitations dominate. The use of parallel processing may alleviate some problems but such techniques are not yet fully developed. Apart from this multimedia database consume a lot of processing time as well as bandwidth.

5. **Queries and retrieval –**For multimedia data like images, video, audio accessing data through query opens up many issues like efficient query formulation, query execution and optimization which need to be worked upon.

Are as where multimedia database is applied are:

- **Documents and record management:** Industries and businesses that keep detailed records and variety of documents. Example: Insurance claim record.
- **Knowledge dissemination:** Multimedia database is a very effective tool for knowledge dissemination in terms of providing several resources. Example: Electronic books.
- **Education and training:** Computer-aided learning materials can be designed using multimedia sources which are nowadays very popular sources of learning. Example: Digital libraries.
- Marketing, advertising, retailing, entertainment and travel. Example: a virtual tour of cities.
- **Real-time control and monitoring:** Coupled with active database technology, multimedia presentation of information can be very effective means for monitoring and controlling complex tasks Example: Manufacturing operation control.

DATABASE SUPPORT IN WEB APPLICATIONS IN JAVA

In Java, there are several options for database support in web applications.
Some popular ones include**:**

Java Database Connectivity (JDBC): JDBC is the standard API for connecting to relational databases in Java. It provides a set of classes and methods that allow developers to perform database operations, such as querying and updating data. JDBC enables you to connect to various databases, including MySQL, Oracle, and Post greSQL, among others.

Object-Relational Mapping (ORM) frameworks: ORM frameworks like Hibernate and Eclipse Link offer a higher-level abstraction over JDBC. They allow you to map Java objects to database tables, making it easier to perform database operations. ORM frameworks handle the generation of SQL queries and provide additional features like caching and transaction management.

Java Persistence API (JPA): JPA is a specification that defines a standard way to map Java objects to relational databases. It is a part of the Java EE (Enterprise Edition) platform and provides a set of interfaces and annotations to simplify database operations. JPA implementations, such as Hibernate and Eclipse Link, offer ORM capabilities along with JPA compliance.

Spring Data: Spring Data is a module within the Spring Framework that provides easy database access and manipulation. It offers a set of high-level abstractions for working with databases and offers consistent APIs across different data storage technologies, such as relational databases, No SQL databases, and more.

These are just a few examples of how Java web applications can be supported by databases. The choice of database support depends on the specific requirements and preferences of the project.

Java Web Application

Java Web Application is used to create dynamic websites. Java provides support for web application through **Servlets** and **JSPs**. We can create a website with static HTML pages but when we want information to be dynamic, we need web application.

The aim of this article is to provide basic details of different components in Web Application and how can we use Servlet and JSP to create our first java web application.

Java Web Application is used to create dynamic websites. Java provides support for web application through **Servlets** and **JSPs**. We can create a website with static HTML pages but when we want information to be dynamic, we need web application.

Java Web Application

The aim of this article is to provide basic details of different components in Web Application and how can we use Servlet and JSP to create our first java web application.

Web Server and Client

Web Server is a software that can process the client request and send the response back to the client. For example, Apache is one of the most widely used web servers. Web Server runs on some physical machine and listens to client request on a specific port.

A web client is a software that helps in communicating with the server. Some of the most widely used web clients are Firefox, Google Chrome, Safari, etc.

When we request something from the server (through URL), the web client takes care of creating a request and sending it to the server and then parsing the server response and present it to the user.

HTML and HTTP

Web Server and Web Client are two separate softwares, so there should be some common language for communication. HTML is the common language between server and client and stands for **H**yper**T**ext **M**arkup **L**anguage. Web server and client needs a common communication protocol, HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) is the communication protocol between server and client. HTTP runs on top of TCP/IP communication protocol. Some of the important parts of the HTTP Request are:

- **HTTP Method** - action to be performed, usually GET, POST, PUT etc.
- **URL** - Page to access
- **Form Parameters** - similar to arguments in a java method, for example user,password details from login page.

Sample HTTP Request:

```
GET /FirstServletProject/jsps/hello.jsp HTTP/1.1
Host: localhost:8080
Cache-Control: no-cache
```

Some of the important parts of HTTP Response are:

- **Status Code** - an integer to indicate whether the request was success or not. Some of the well-known status codes are 200 for success, 404 for Not Found and 403 for Access Forbidden.
- **Content Type** - text, html, image, pdf etc. Also known as MIME type
-
- **Content** - actual data that is rendered by client and shown to user.

Sample HTTP Response:

```
200 OK
Date: Wed, 07 Aug 2013 19:55:50 GMT
Server: Apache-Coyote/1.1
Content-Length: 309
Content-Type: text/html;charset=US-ASCII

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Hello</title>
</head>
<body>
<h2>Hi There!</h2>
<br>
<h3>Date=Wed Aug 07 12:57:55 PDT 2013
</h3>
</body>
</html>
```

MIME Type or Content Type: If you see above sample HTTP response header, it contains tag "Content-Type". It's also called MIME type and server sends it to the client to let them know the kind of data it's sending. It helps the client in rendering the data for the user. Some of the most used mime types are text/html, text/xml, application/xml etc.

Understanding URL

URL is the acronym of Universal Resource Locator and it's used to locate the server and resource. Every resource on the web has its own unique address. Let's see parts of the URL with an

example. **https://localhost:8080/FirstServletProject/jsps/hello.jsp** **https://** - This is the first part of URL and provides the communication protocol to be used in server-client communication. **localhost** - The unique address of the server, most of the times it's the hostname of the server that maps to unique IP address.

Sometimes multiple hostnames point to same IP addresses and web server virtual host takes care of sending a request to the particular server instance. **8080** - This is the port on which server is listening, it's optional and if we don't provide it in URL then request goes to the default port of the protocol.

Port numbers 0 to 1023 are reserved ports for well-known services, for example, 80 for HTTP, 443 for HTTPS, 21 for FTP, etc. **FirstServletProject/jsps/hello.jsp** - Resource requested from server. It can be static html, pdf, JSP, servlets, PHP etc.

Web servers are good for static contents HTML pages but they don't know how to generate dynamic content or how to save data into databases, so we need another tool that we can use to generate dynamic content.

There are several programming languages for dynamic content like PHP, Python, Ruby on Rails, Java Servlets and JSPs. Java Servlet and JSPs are server-side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.

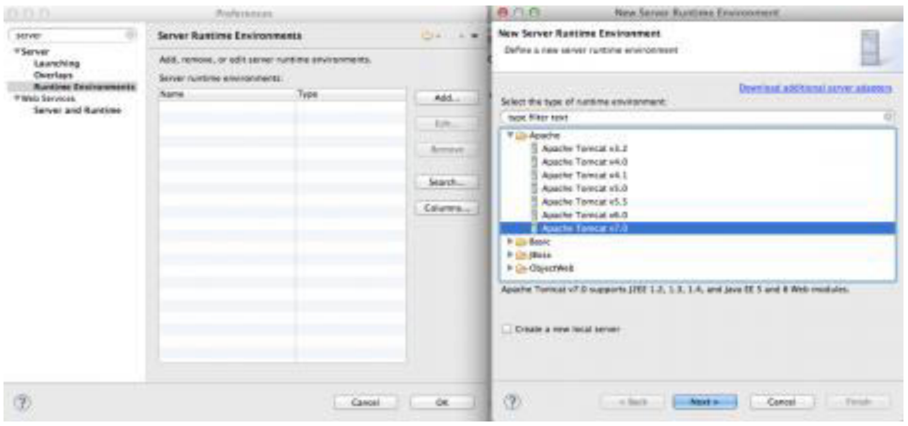
Java Web Development

First Web Application with Servlet and JSP

We will use “Eclipse IDE for Java EE Developers” for creating our first servlet application. Since servlet is a server-side technology, we will need a web container that supports Servlet technology, so we will use the Apache Tomcat server.

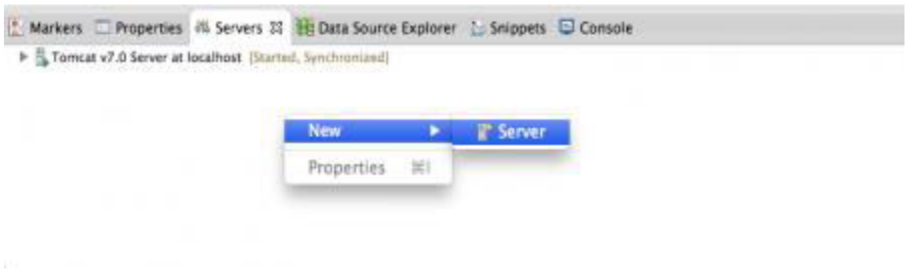
It’s very easy to set up and I am leaving that part to yourself. For ease of development, we can add configure Tomcat with Eclipse, it helps in easy deployment and running applications.

Go to Eclipse Preference and select Server Runtime Environments and select the version of your tomcat server, mine is Tomcat 7.



Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

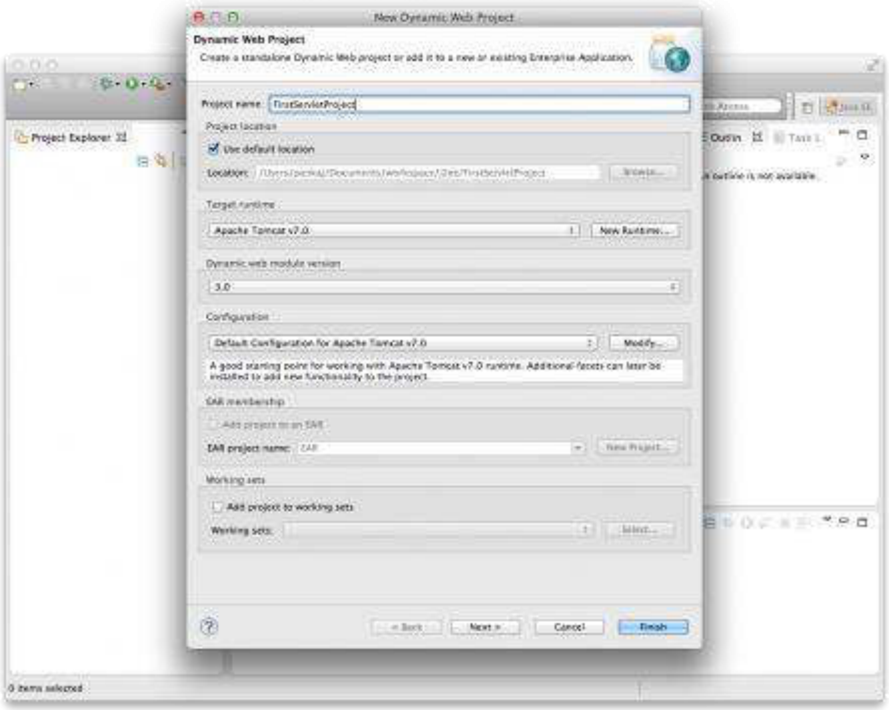
Provide the apache tomcat directory location and JRE information to add the runtime environment. Now go to the Servers view and create a new server like below image pointing to the above-added runtime environment.



Note: If Servers tab is not visible, then you can select Window > Show View > Servers so that it will be visible in Eclipse window.

Try stopping and starting the server to make sure it’s working fine. If you have already started the server from the terminal, then you will have to stop it from the terminal and then start it from Eclipse else it won’t work perfectly. Now we are ready with our setup to create the first servlet and run it on tomcat server.

Select File > New > Dynamic Web Project and use below image to provide runtime as the server we added in last step and module version as 3.0 to create our servlet using



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: FirstServletProject

Project location

☑ Use default location

Location: /Users/pankaj/Documents/workspace/J2ee/FirstServletProject

Target runtime: Apache Tomcat v7.0

Dynamic web module version: 3.0

Configuration: Default Configuration for Apache Tomcat v7.0

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

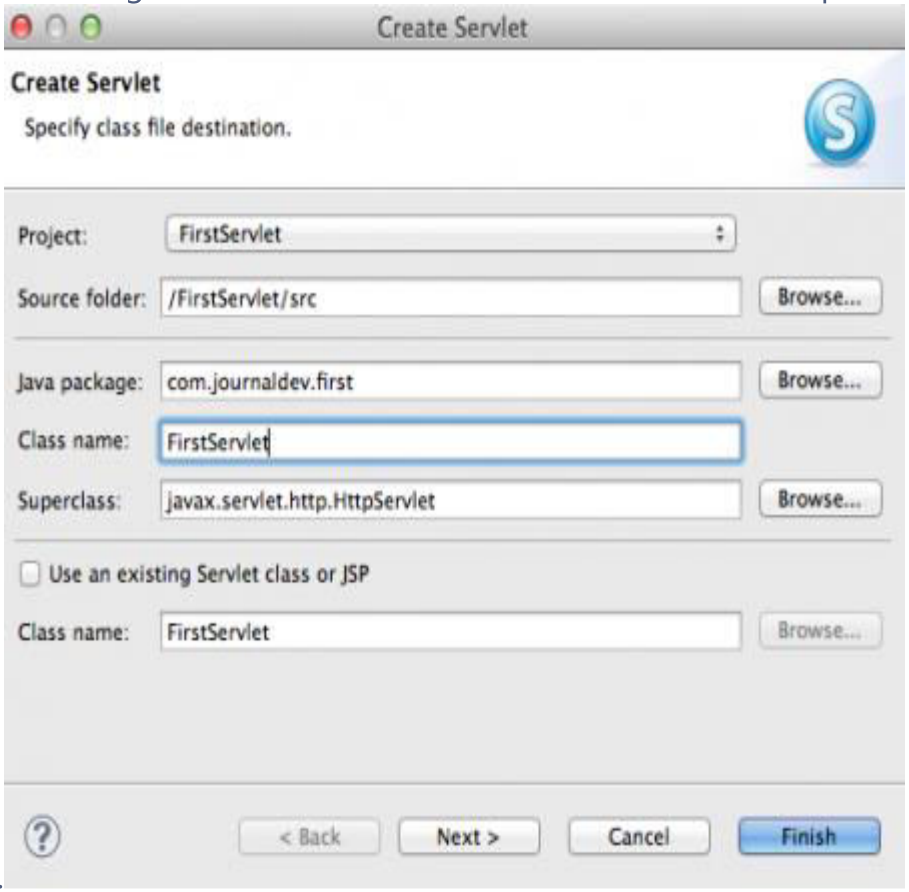
EAR project name: EAR

Working sets

☐ Add project to working sets

Working sets:

Servlet 3.0 specs. You can directly click the Finish button to create the project or you can click on Next buttons to check for other options. Now select File > New > Servlet and use below image to create our first servlet. Again we can click finish or we can check other options through



Create Servlet

Create Servlet

Specify class file destination.

Project: FirstServlet

Source folder: /FirstServlet/src

Java package: com.journaldev.first

Class name: FirstServlet

Superclass: javax.servlet.http.HttpServlet

☐ Use an existing Servlet class or JSP

Class name: FirstServlet

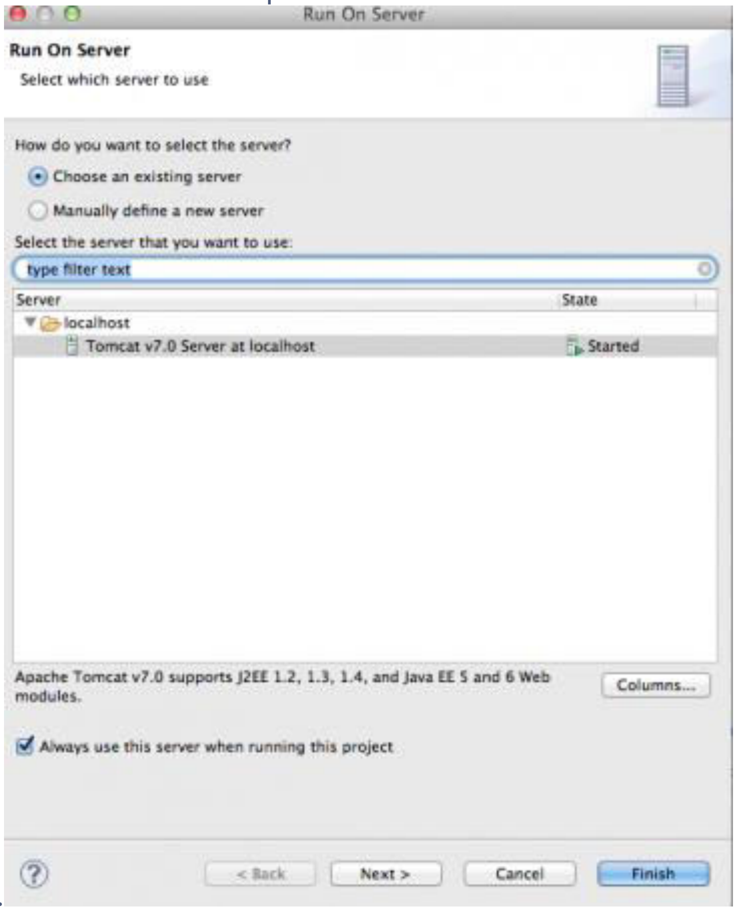
the next button.

When we click on the Finish button, it generates our Servlet skeleton code, so we don’t need to type in all the different methods and imports in servlet and saves us time. Now we will add some HTML with dynamic data code in **doGet()** method that will be invoked for HTTP GET request. Our first servlet looks like below.

```
package com.journaldev.first;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class FirstServlet
 */
@WebServlet(description = "My First Servlet", urlPatterns = { "/FirstServlet" ,
"/FirstServlet.do"}, initParams =
{@WebInitParam(name="id",value="1"),@WebInitParam(name="name",value="pankaj")})
public class FirstServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	public static final String HTML_START="<html><body>";
	public static final String HTML_END="</body></html>";

    /**
     * @see HttpServlet#HttpServlet()
     */
    public FirstServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		PrintWriter out = response.getWriter();
		Date date = new Date();
		out.println(HTML_START + "<h2>Hi 
There!</h2><br/><h3>Date="+date +"</h3>"+HTML_END);
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse 
response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse 
response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Before Servlet 3, we need to provide the url pattern information in web application deployment descriptor but servlet 3.0 uses **java annotations** that is easy to understand and chances of errors are less.

Now chose Run > Run on Server option from servlet editor window and use below images for the options.



Run On Server

Run On Server

Select which server to use

How do you want to select the server?

[✓ Choose an existing server]

Manually define a new server

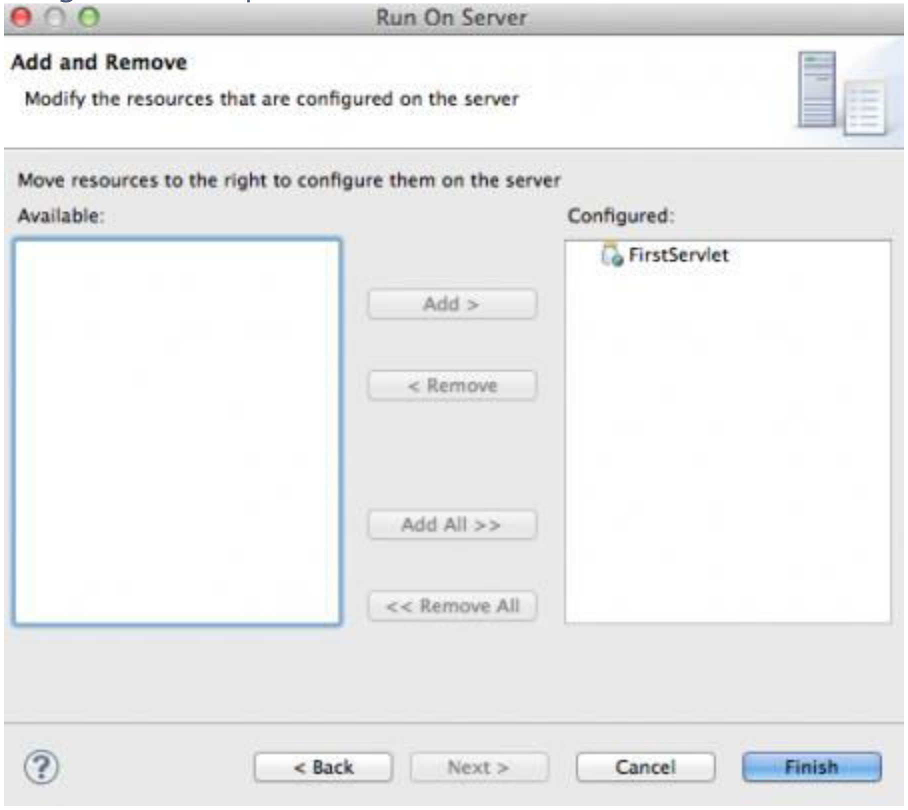
Select the server that you want to use:

type filter text

Server	State
localhost	
Tomcat v7.0 Server at localhost	Started

Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

[✓] Always use this server when running this project



Run On Server

Add and Remove

Modify the resources that are configured on the server

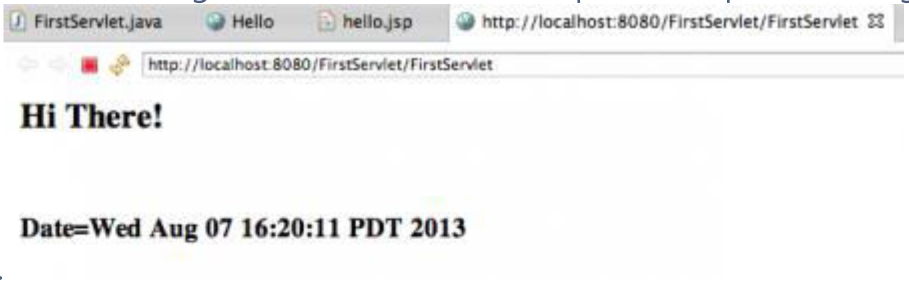
Move resources to the right to configure them on the server

Available:

Configured:

FirstServlet

After clicking finish, the browser will open in Eclipse and we get following HTML



Hi There!

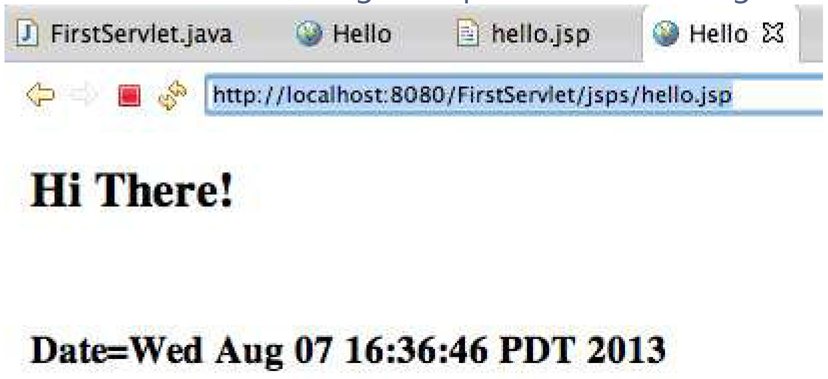
Date=Wed Aug 07 16:20:11 PDT 2013

page.

You can refresh it to check that Date is dynamic and keeps on changing, you can open it outside of Eclipse also in any other browser. So servlet is used to generate HTML and send it in response, if you will look into the doGet() implementation, we are actually creating an HTML document as writing it in response PrintWriter object and we are adding dynamic information where we need it. It's good for a start but if the response is huge with a lot of dynamic data, it's error-prone and hard to read and maintain. This is the primary reason for the introduction of JSPs. JSP is also server-side technology and it's like HTML with additional features to add dynamic content where we need it. JSPs are good for presentation because it's easy to write because it's like HTML. Here is our first JSP program that does the same thing as the above servlet.

```
<%@page import="java.util.Date"%>
<%@ page language="java" contentType="text/html; charset=US-ASCII"
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Hello</title>
</head>
<body>
<h2>Hi There!</h2>
<br>
<h3>Date=<%= new Date() %>
</h3>
</body>
</html>
```

If we run above JSP, we get output like below image.

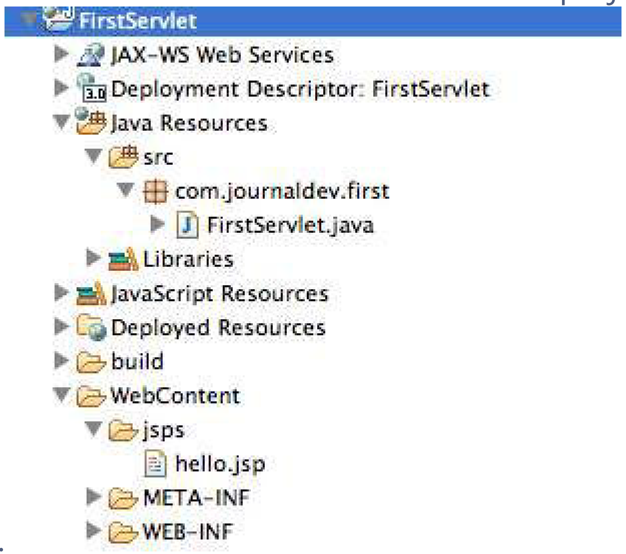


http://localhost:8080/FirstServlet/jsps/hello.jsp

Hi There!

Date=Wed Aug 07 16:36:46 PDT 2013

The final project hierarchy looks like below image in Eclipse.



```
FirstServlet
  JAX-WS Web Services
  Deployment Descriptor: FirstServlet
  Java Resources
    src
      com.journaldev.first
        FirstServlet.java
    Libraries
  JavaScript Resources
  Deployed Resources
  build
  WebContent
    jsps
      hello.jsp
    META-INF
    WEB-INF
```

We will look into Servlets and JSPs in more detail in future posts but before concluding this post, we should have a good understanding of some of the aspects of Java web applications.

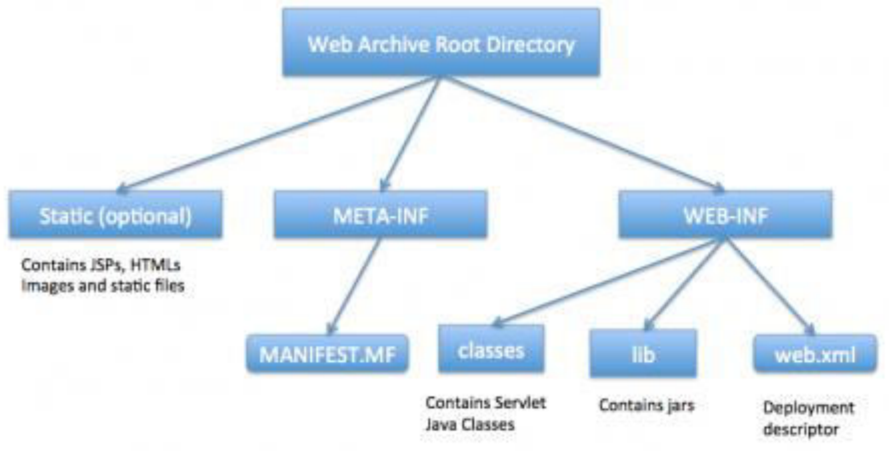
Web Container

Tomcat is a web container, when a request is made from Client to web server, it passes the request to web container and it’s web container job to find the correct resource to handle the request (servlet or JSP) and then use the response from the resource to generate the response and provide it to web server. Then the webserver sends the response back to the client. When web container gets the request and if it’s for servlet then container creates two Objects HTTPServletRequest and HTTPServletResponse. Then it finds the correct servlet based on the URL and creates a thread for the request. Then it invokes the servlet service () method and based on the HTTP method service () method invokes do Get () or do Post () methods. Servlet methods generate the dynamic page and write it to the response. Once servlet thread is complete, the container converts the response to HTTP response and send it back to the client. Some of the important work done by web container are:

- **Communication Support** - Container provides easy way of communication between web server and the servlets and JSPs. Because of the container, we don't need to build a server socket to listen for any request from the webserver, parse the request and generate a response. All these important and complex tasks are done by container and all we need to focus is on our business logic for our applications.
- **Lifecycle and Resource Management** - Container takes care of managing the life cycle of servlet. The container takes care of loading the servlets into memory, initializing servlets, invoking servlet methods and destroying them. The container also provides utility like JNDI for resource pooling and management.
- **Multithreading Support** - Container creates a new thread for every request to the servlet and when it's processed the thread dies. So servlets are not initialized for each request and save time and memory.
- **JSP Support** - JSPs doesn't look like normal java classes and web container provides support for JSP. Every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets.
- **Miscellaneous Task** - Web container manages the resource pool, does memory optimizations, run garbage collector, and provides security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes our life easier.

Web Application Directory Structure

Java Web Applications are packaged as Web Archive (WAR) and it has a defined structure. You can export above dynamic web project as WAR file and unzip it to check the hierarchy. It will be something like below image.


Web Archive Root Directory
Static (optional)
Contains JSPs, HTMLs Images and static files
META-INF
WEB-INF
MANIFEST.MF
classes
Contains Servlet Java Classes
lib
Contains jars
web.xml
Deployment descriptor

Deployment Descriptor

Web.xml file is the deployment descriptor of the web application and contains a mapping for servlets (prior to 3.0), welcome pages, security configurations, session timeout settings, etc.

UNIT-IV

SERVLETS

JAVA SERVLETS:

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

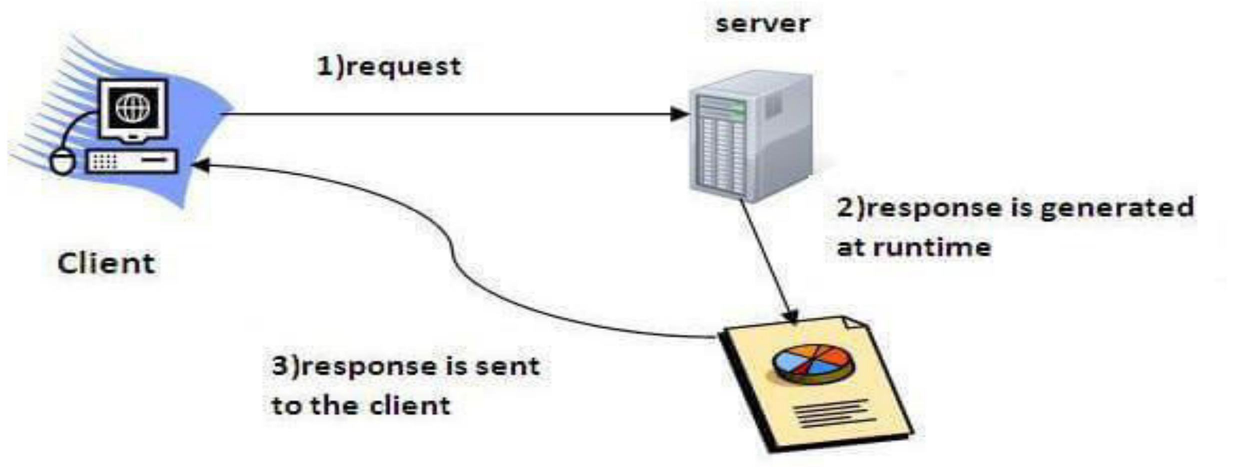
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, Generic Servlet, Http Servlet, Servlet Request, Servlet Response, etc.

Servlet

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

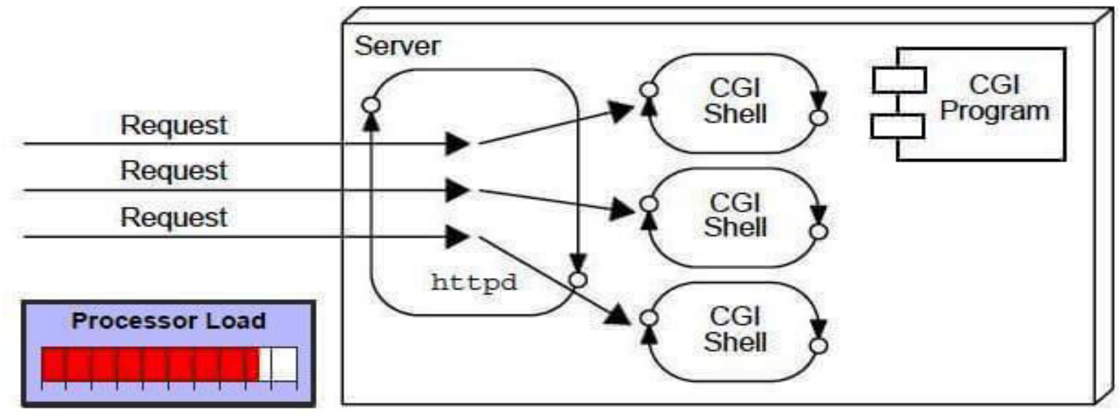

server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

WEB APPLICATION

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

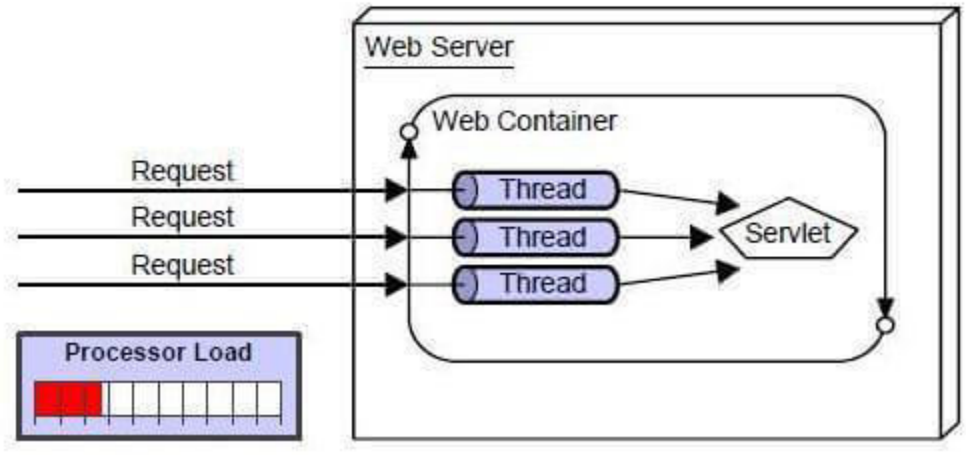
There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet

There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.



CGI: COMPUTER GENERATED IMAGERY

CGI stands for Computer Generated Imagery. It is an application of computer graphics (imaging software) that is used to create realistic-looking (three-dimensional) images, still and animated visual content, anatomical modeling, architectural design, video game art, special effects in movies and electronic media, etc. In short, it allows you to create characters and motions that look real and which may not be created using other methods.

This technique manipulates the environment and creates photorealistic images for print and electronic media such as movies, videos, games, etc. As CGI visuals are more cost-effective than traditional photographic ones, they are widely used throughout the world. A single artist can produce content with CGI without using actors, set pieces, or props.

CGI is created with the help of wireframe models. The features like reflection and illumination can be assigned to the wireframes. These features can be modified as per the requirement of the image and video in order to make them look real. The quality of the visual effects produced by CGI is higher and controllable as compared to physical ones, such as creating miniatures for shots, hiring extras for crowd scenes, etc.

CGI Works:

First, the artists create computer-generated graphics, and then to make graphics look real the texture, lighting, and color are adjusted. These adjustments make the animation look real and not cartoonish. In live-action films, the graphics are mixed with the previously filmed scenes. In this case, the lighting on the graphics must match the lighting from the scene to make the finished product seamless.

The CGI was used for the first in a movie in 1973; Michael Crichton's "West world". After a few years, it was used in the movie "Star Wars." In 1993, it was used in the movie "Jurassic Park". There are many other movies that make good use of CGI, such as Avatar, Lord of the Rings, Inception, Finding Nemo, The Matrix, and more.

History

The earliest examples of computer-generated imagery may be found in the 1950s when mechanical computers were used to design patterns on animation cels that were then incorporated into a feature picture. Vertigo, directed by Alfred Hitchcock, was the first movie to employ CGI (1958).

Even though Alfred may have started things off early with some 2D trickery, it was not until Edwin Cat mull and Fred Parke's 1972 computer-animated short film A Computer Animated Hand that 3D computer graphics were officially exposed to the world. To do this, Edwin drew 350 triangles and polygons in ink on his hand, which were then transformed to digital form and laboriously animated utilizing information in a 3D animation programme that Catmull himself developed.

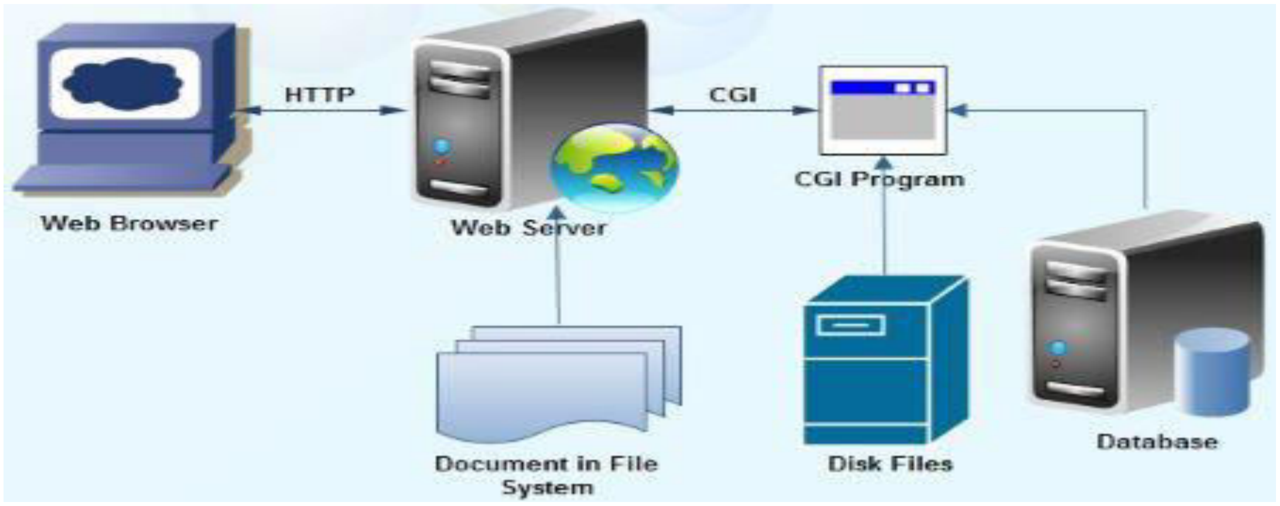
A few years later, with Hollywood's support, CGI made yet another advancement. West world flexed its biceps in 1973 by releasing the first 2D CGI scene showcasing "Gunslinger vision," a theory of how robots could see. A sequel was produced since the first film was so popular.

(ii) CGI: Common Gateway Interface

CGI stands for Common Gateway Interface. It is a technology that enables a web browser to submit forms and connect to programs over a web server. It is the best way for a web server to send forms and connect to programs on the server. CGI can also be described as a set of standards or rules where a program or script can send data back to the web server where it can be processed.

So, it is an interface for running executable via a web server. In general, it means taking an HTTP request and passing it to an application in order to deliver a dynamically generated HTML page back to a browser. However, any program that can run on a web server is usable as a CGI script. Generally, CGI programs are used to generate pages dynamically or to perform some other action when someone fills out an HTML form and clicks the submit button. CGI applications can be written in any programming language, some of which are Perl, PHP, and Python.

CGI works



The browser sends a URL that causes the AOL server to use CGI to run a program. The browser runs on a client machine and exchanges information with a Web server using the Hyper Text Transfer Protocol or HTTP.

Depending on the type of request from the browser, the web server may provide a document from its own document directory or executes a CGI program which means it passes the input from the reader (browser) to the program and output from the program back to the reader (browser). Thus, CGI works as a gateway between the AOL server and the program you write.

The steps involved in creating a dynamic HTML document on the fly through CGI are as follows:

1. The client sends an HTTP request through a URL.
2. From the URL, the Web server decides that it should activate the gateway program listed in the URL and send any parameters passed via the URL to that program.
3. The gateway program processes the information and returns HTML text to the Web server. The Web server adds a MIME header and sends the HTML text to the Web browser.
4. The web browser renders the document received from the web server.

Qualities of CGI

The following list includes some of the advantages of CGI:

- It is a very well-supported and defined standard.
- Typically, CGI scripts are written in Perl, C, or even just a straightforward shell script.
- The technology called CGI communicates with HTML.
- CGI is currently the fastest way to build a counter; hence it should be used.
- The CGI standard is typically the one that works best with modern browsers.

Benefits of CGI:

- Currently, CGI is easier to use than Java for performing complex operations.
- Using pre-written code is usually simpler than writing your own.
- As long as they adhere to the definition, CGI allows programmes to be created in any language and on any platform.
- There are many CGI-based counters and CGI programmes that can carry out basic functions.

Disadvantages of CGI

The following list includes some of the disadvantages of CGI:

- Because programmers must be loaded into memory for each page load in Common Gateway Interface, overhead is incurred.
- In general, it is difficult to cache data in memory between page loads.
- A substantial amount of existing code is written in Perl.
- Processing time for CGI is high.

A simple java Servlet

A Servlet is a Java program that runs on a web server and processes HTTP requests from web clients, such as web browsers. When a client sends an HTTP request to a web server, the server passes the request to a servlet that is capable of handling that type of request.

Once the servlet receives the request, it can perform a variety of actions, such as accessing a database, processing form data, or generating dynamic web content. Once the servlet has completed its processing, it returns a response to the client in the form of an HTTP response.

Servlets are designed to be highly flexible and configurable. They can be configured to handle different types of HTTP requests, such as GET, POST, and

PUT requests. They can also be configured to handle different types of data, such as HTML, XML, and JSON.

Creating a Servlet Program in Java

Let's take a closer look at the steps involved in creating a basic Servlet program in Java:

Step 1 – Set up your development environment
before you can create a Servlet program in Java, you'll need to set up your development environment. This typically involves installing the Java Development Kit (JDK) and a web server such as Apache Tomcat.

Step 2 – Create a new Java class
once your development environment is set up, you can create a new Java class for your servlet. This class should extend the HttpServlet class, which is part of the Java Servlet API.

Here's an example of a basic servlet class:

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;


public class MyServlet extends HttpServlet {


}
```

This class defines a new servlet called MyServlet.

Step 3 – Override the doGet() or doPost() method
The doGet() and doPost() methods are the two main methods used by servlets to handle HTTP requests. To create a basic servlet program, you'll need to override one or both of these methods in your servlet class.

Here's an example of a basic doGet() method that sends a simple HTTP response:

```
 public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

     PrintWriter out = response.getWriter();

     out.println("Hello, World!");

 }
```

This method sends a simple "Hello, World!" message back to the client as an HTTP response.

Deploying a Servlet

Once we have created a Servlet program in Java, we need to deploy it on a web server to make it accessible over the internet. To deploy a Servlet program in Java, you'll need to follow these basic steps:

Step 1 – Compile the Servlet program: Before you can deploy your Servlet program, you'll need to compile the Java source code into byte code. This can be done using the Java compiler, which is part of the Java Development Kit (JDK). The compiled byte code should be packaged into a .war (Web Archive) file.

Step 2 – Set up a web server: A web server such as Apache Tomcat is required to deploy a servlet program. If you don't already have a web server set up, you can download and install Tomcat from the Apache website.

Step 3 – Deploy the servlet program: Once you have a web server set up, you can deploy the servlet program by copying the .war file to the web apps directory of the Tomcat installation. The web apps directory is located in the Tomcat installation directory.

Step 4 – Start the web server: After you have deployed the .war file, you'll need to start the web server to make the servlet program available. This can be done by running the startup script (startup.bat on Windows, startup.sh on UNIX) located in the bin directory of the Tomcat installation.

Step 5 – Test the servlet program: Once the web server is running, you can test the servlet program by accessing it in a web browser. The URL for the servlet program will depend on the name of the .war file and the name of the servlet class.

For example, if you have deployed a servlet program called "MyServlet" in a .war file called "myservlet.war", the URL to access the servlet might be:

http://localhost:8080/myservlet/MyServlet

In this example, "localhost" is the name of the machine running the web server, "8080" is the default port for Tomcat, "myservlet" is the name of the .war file, and "MyServlet" is the name of the servlet class.

In other words, deploying a servlet program in Java involves compiling the Java source code, setting up a web server such as Apache Tomcat, deploying the .war file to the webapps directory, starting the web server, and testing the servlet program by accessing it in a web browser. Once deployed, the servlet program can handle HTTP requests and generate dynamic web content.

Conclusion

In this article, we have explored the basics of writing a Servlet program in Java. We have seen how to create a Servlet class that handles HTTP requests and generates responses, and how to deploy our Servlet program on a web server. Servlets are a powerful tool for creating web applications in Java, and they provide a standard interface for communicating with web servers.

The Anatomy of a Servlet

As you just saw in the HelloWorldServlet, a *servlet* is a Java class that implements a few important methods. You can choose to implement these methods yourself or create a subclass of an existing servlet class that already implements them. The Servlet interface defines the methods that are required for a Java class to become a servlet. The interface definition is shown in Listing 3.2.

Listing 3.2 The Definition of the Servlet Interface

```
package javax.servlet;



public interface Servlet

{

 public void destroy();

 public ServletConfig getServletConfig();

 public String getServletInfo();

 public void init(ServletConfig config)

  throws ServletException;

 public void service(ServletRequest request,

  ServletResponse response)
  throws ServletException, java.io.IOException;

}
```

Most of the time, you will create a servlet by subclassing
either GenericServlet or HttpServlet. Both of these classes implement the Servlet interface,
but they provide a few handy features that make them preferable to implementing
the Servlet interface yourself.

The service Method
The heart of any servlet is the service method. As you just learned, the servlet engine calls
the service method to handle each request from a browser, passing in an object
containing both information about the request that invoked the servlet and information
about sending back a response. The service method is the only method that a servlet is
actually required to implement. The service method is declared this way:

```
 public void service(ServletRequest request,

  ServletResponse response)

  throws java.io.IOException
```

The init Method
Many times, a servlet needs to perform some initialization one time before it begins to
handle requests. The init method in a servlet is called just after the servlet is first loaded,
but before it begins to handle requests. Listing 3.3 shows a simple init method that
initializes a database connection.
Listing 3.3 init Method from JDBCServlet.java

```
protected Connection conn;



public void init()
```

```
{

 try

 {

// Make sure the JdbcOdbcDriver class is loaded

  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");



// Try to connect to a database via ODBC

  conn = DriverManager.getConnection(

   "jdbc:odbc:usingjsp");

 }

 catch (Exception exc)

 {

// If there's an error, use the servlet logging API

  getServletContext().log(

   "Error making JDBC connection: ", exc);

 }

}
```

Notice that the init method in Listing 3.3 does not take a parameter like the init method in the Servlet interface. One of the convenient features of the GenericServlet and HttpServlet classes is that they have an alternate version of init that doesn't take any parameters. In case you're wondering why it even matters, the init method in the Servlet interface takes a ServletConfig object as a parameter. The servlet is then responsible for keeping track of the ServletConfig object. The GenericServlet and HttpServlet classes perform this housekeeping chore and then provide the parameter-less init method for you to do any servlet-specific initialization.

If you override the init(ServletConfig config) method of GenericServlet or HttpServlet, make sure you call super.init(config) as the first statement in your init method so that the housekeeping will still be performed.

The destroy Method

Sometimes, the servlet engine decides that it doesn't need to keep your servlet loaded anymore. This could happen automatically, or as the result of you deactivating the servlet from an administration tool. Before the servlet engine unloads your servlet, it calls the destroy method to enable the servlet to perform any necessary cleanup. The cleanup usually involves closing database connections, open files, and network connections. Listing 3.4 shows the destroy method that is a companion to the init method in Listing 3.3.

Listing 3.4 destroy Method from JDBCServlet.java

```
public void destroy()

{

 try

 {

// Only try to close the connection if it's non-null

  if (conn != null)

  {
   conn.close();

  }

 }

 catch (SQLException exc)

 {

// If there's an error, use the servlet logging API

  getServletContext().log(

   "Error closing JDBC connection: ", exc);

 }

}
```

The getServletInfo and getServletConfig Methods
If you are subclassing GenericServlet or HttpServlet, you probably won't need to override the getServletInfo or getServletConfig methods. The Servlet API documentation recommends that you return information such as the author, version, and copyright from the getServletInfo method. Although there is no specific format for the string returned by the method, you should return only plain text, without any HTML or XML tags embedded within it.

The getServletConfig method returns the ServletConfig object that was passed to the servlet in the init method. Unless you are keeping track of the config object yourself, your best bet is to leave this method alone and let the superclass handle it.

Reading Form Data :

There are three methods of servlet to handle the form data. These are listed below -

Servlet Session Examples

Servlet Session Examples

- **getParameter(String name) -** It returns the value of a request parameter as a String. If there is no parameter, return null.
  This method is used when parameter has single value.
- **getParameterNames() -** it returns an enumeration of String objects containing complete list of all parameters
  in the given request.
- **getParameterValues() -** It returns array of String objects containing all the values of the requested parameter.
  If there is no parameter, return null.

Example:

Here we are taking example of ***getParameter()***. **form.jsp** is our jsp page in which we are designing form having two fields name and location.

In servlet **GetParameterExample.java** we are getting name and location by using ***getParameter()*** method.

form.jsp -

```
<html>

<body>

<form method="get" action="getParameterExample">

<table align="center">

<tr>

<td>Name : </td>

<td><input type="text" name="name"/></td>

</tr>

<tr>

<td>Location : </td>

<td><input type="text" name="location"/></td>

</tr>

<tr>


<td><input type="submit" name="Submit" value="Submit"/></td>
```

```
</tr>
</table>
</form>
</body>
</html>
```

GetParameterExample.java -

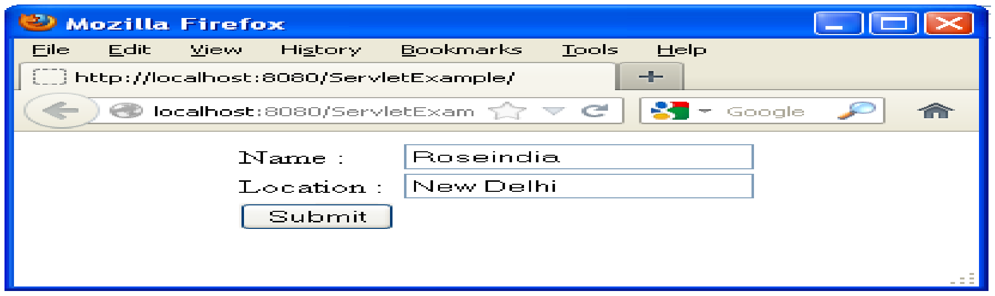
```
package net.roseindia;


import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class GetParameterExample extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String name = request.getParameter("name");
String location = request.getParameter("location");


out.println("<b><font color='purple'>Your Name :</font>" + name + "</b></br>");
out.println("<b><font color='purple'>Your Location :</font>" + location+"</b>");



}
}
```

web.xml -

```
<?xml version="1.0" encoding="UTF-8"?>
```

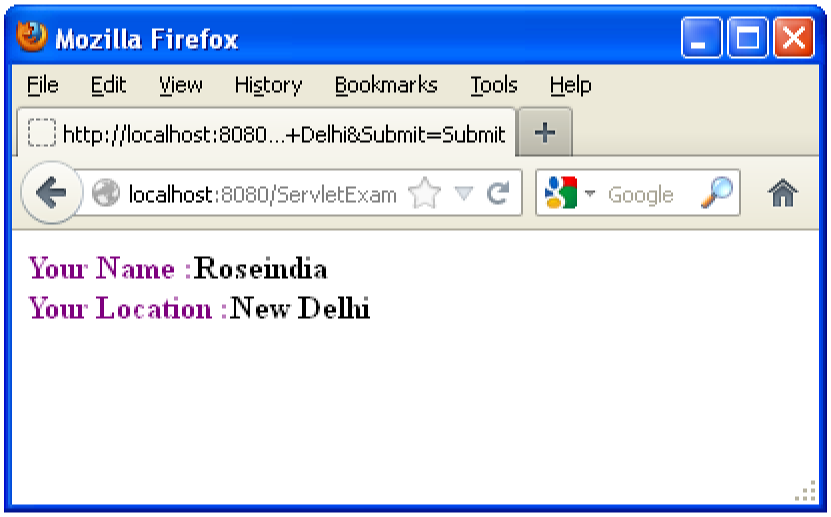
```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.
sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/java
ee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>ServletExample</display-name>


<!--GetParameter servlet mapping-->

<servlet>

<servlet-name>GetParameterExample</servlet-name>

<servlet-class>net.roseindia.GetParameterExample</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>GetParameterExample</servlet-name>

<url-pattern>/getParameterExample</url-pattern>

</servlet-mapping>


<welcome-file-list>

<welcome-file>form.jsp</welcome-file>


</welcome-file-list>

</web-app>
```

Output :


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/ServletExample/
localhost:8080/ServletExam
Google

Name : Roseindia
Location : New Delhi
Submit

When you click on submit you will get following result page.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080...+Delhi&Submit=Submit
localhost:8080/ServletExam
Google

Your Name :Roseindia
Your Location :New Delhi

An Overview of Request Headers

When an HTTP client (e.g. a browser) sends a request, it is required to supply a request line (usually GET or POST). If it wants to, it can also send a number of headers, all of which are optional except for Content-Length, which is required only for POST requests. Here are the most common headers:

- Accept The MIME types the browser prefers.
- Accept-Charset The character set the browser expects.
- Accept-Encoding The types of data encodings (such as gzip) the browser knows how to decode. Servlets can explicitly check for gzip support and return gzipped HTML pages to browsers that support them, setting the Content-Encoding response header to indicate that they are gzipped. In many cases, this can reduce page download times by a factor of five or ten.
- Accept-Language The language the browser is expecting, in case the server has versions in more than one language.
- Authorization Authorization info, usually in response to a WWW-Authenticate header from the server.
- Connection Use persistent connection? If a servlet gets a Keep-Alive value here, or gets a request line indicating HTTP 1.1 (where persistent connections are the default), it may be able to take advantage of persistent connections, saving significant time for Web pages that include several small pieces (images or applet classes). To do this, it needs to send a Content-Length header in the *response*, which is most easily accomplished by writing into a ByteArrayOutputStream, then looking up the size just before writing it out.
- Content-Length (for POST messages, how much data is attached)

- Cookie (one of the most important headers; see separate section in this tutorial on handling cookies)
- From (email address of requester; only used by Web spiders and other custom clients, not by browsers)
- Host (host and port as listed in the *original* URL)
- If-Modified-Since (only return documents newer than this, otherwise send a 304 "Not Modified" response)
- Pragma (the no-cache value indicates that the server should return a fresh document, even if it is a proxy with a local copy)
- Referer (the URL of the page containing the link the user followed to get to current page)
- User-Agent (type of browser, useful if servlet is returning browser-specific content)
- UA-Pixels, UA-Color, UA-OS, UA-CPU (nonstandard headers sent by some Internet Explorer versions, indicating screen size, color depth, operating system, and cpu type used by the browser's system)

Reading Request Headers from Servlets

Reading headers is very straightforward; just call the getHeader method of the HttpServletRequest, which returns a String if the header was supplied on this request, null otherwise. However, there are a couple of headers that are so commonly used that they have special access methods.

The getCookies method returns the contents of the Cookie header, parsed and stored in an array of Cookie objects. See the separate section of this tutorial on cookies. The getAuthType and getRemoteUser methods break the Authorization header into its component pieces.

The getDateHeader and getIntHeader methods read the specified header and then convert them to Date and int values, respectively.

Rather than looking up one particular header, you can use the get Header Names to get an Enumeration of all header names received on this particular request.

Finally, in addition to looking up the request headers, you can get information on the main request line itself. The get Method method returns the main request method (normally GET or POST, but things like HEAD, PUT, and DELETE are possible).

The get Request URI method returns the URI (the part of the URL that came after the host and port, but before the form data). The get Request Protocol returns the third part of the request line, which is generally "HTTP/1.0" or "HTTP/1.1".

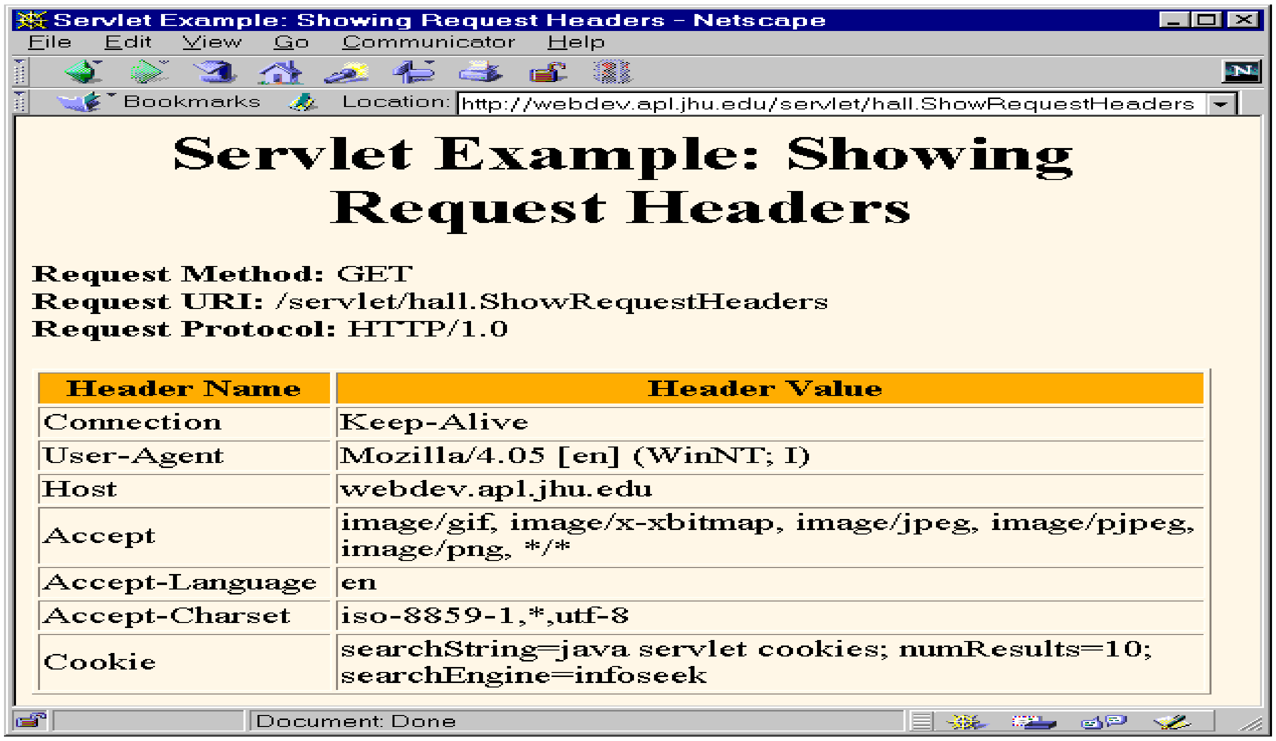
Example: Printing all Headers

Here's a servlet that simply creates a table of all the headers it receives, along with their associated values. It also prints out the three components of the main request line (method, URI, and protocol).

```
package hall;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class ShowRequestHeaders extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Servlet Example: Showing Request Headers";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<B>Request Method: </B>" +
                request.getMethod() + "<BR>\n" +
                "<B>Request URI: </B>" +
                request.getRequestURI() + "<BR>\n" +
                "<B>Request Protocol: </B>" +
                request.getProtocol() + "<BR><BR>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>Header Name<TH>Header Value");

    Enumeration headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()) {
      String headerName = (String)headerNames.nextElement();
      out.println("<TR><TD>" + headerName);
      out.println("    <TD>" + request.getHeader(headerName));
    }
    out.println("</TABLE>\n</BODY></HTML>");
  }

  public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
          throws ServletException, IOException {
        doGet(request, response);
      }
    }
```

3.2 ShowRequestHeaders Output

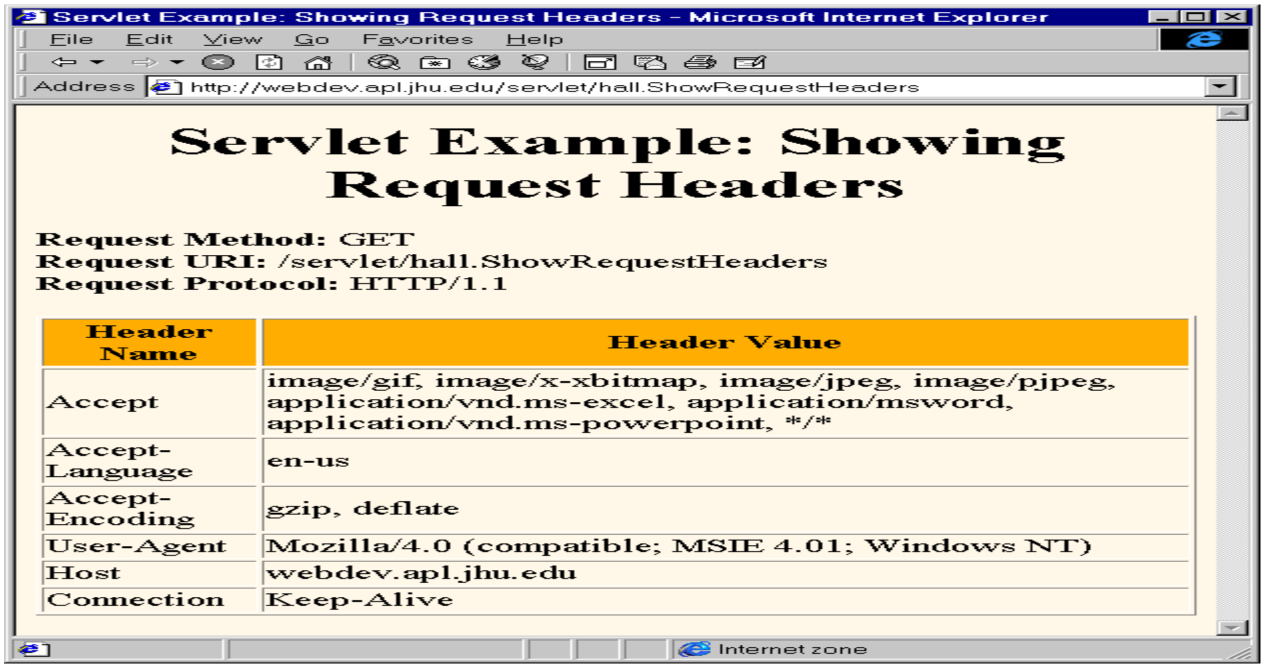
Here are the results of two typical requests, one from Netscape and one from Internet Explorer.


Servlet Example: Showing Request Headers - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://webdev.apl.jhu.edu/servlet/hall.ShowRequestHeaders
Document: Done

Servlet Example: Showing Request Headers

Request Method: GET
Request URI: /servlet/hall.ShowRequestHeaders
Request Protocol: HTTP/1.0

Header Name	Header Value
Connection	Keep-Alive
User-Agent	Mozilla/4.05 [en] (WinNT; I)
Host	webdev.apl.jhu.edu
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Language	en
Accept-Charset	iso-8859-1,*,utf-8
Cookie	searchString=java servlet cookies; numResults=10; searchEngine=infoseek


Servlet Example: Showing Request Headers - Microsoft Internet Explorer
File Edit View Go Favorites Help
Address http://webdev.apl.jhu.edu/servlet/hall.ShowRequestHeaders
Internet zone

Servlet Example: Showing Request Headers

Request Method: GET
Request URI: /servlet/hall.ShowRequestHeaders
Request Protocol: HTTP/1.1

Header Name	Header Value
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Language	en-us
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)
Host	webdev.apl.jhu.edu
Connection	Keep-Alive

SENDING DATA TO A CLIENT AND WRITING THE HTTP RESPONSE HEADER

HTTP Response

HTTP Response sent by a server to the client. The response is used to provide the client with the resource it requested. It is also used to inform the client that the action requested has been carried out. It can also inform the client that an error occurred in processing its request.

An HTTP response contains the following things:

1. Status Line
2. Response Header Fields or a series of HTTP headers
3. Message Body

In the request message, each HTTP header is followed by a carriage returns line feed (CRLF). After the last of the HTTP headers, an additional CRLF is used and then begins the message body.

Status Line

In the response message, the status line is the first line. The status line contains three items:

a) HTTP Version Number

It is used to show the HTTP specification to which the server has tried to make the message comply.

Example

1. HTTP-Version = HTTP/1.1

b) Status Code

It is a three-digit number that indicates the result of the request. The first digit defines the class of the response. The last two digits do not have any categorization role. There are five values for the first digit, which are as follows:

Code and Description

1xx: Information

It shows that the request was received and continuing the process.

2xx: Success

It shows that the action was received successfully, understood, and accepted.

3xx: Redirection

It shows that further action must be taken to complete the request.

4xx: Client Error

It shows that the request contains incorrect syntax, or it cannot be fulfilled.

5xx: Server Error

It shows that the server failed to fulfil a valid request.

c) Reason Phrase

It is also known as the status text. It is a human-readable text that summarizes the meaning of the status code.

An example of the response line is as follows:

```
HTTP/1.1 200 OK
```

Here,

- HTTP/1.1 is the HTTP version.
- 200 is the status code.
- OK is the reason phrase.

Response Header Fields

The HTTP Headers for the response of the server contain the information that a client can use to find out more about the response, and about the server that sent it. This information is used to assist the client with displaying the response to a user, with storing the response for the use of future, and with making further requests to the server now or in the future.

```
response-header = Accept-Ranges
                  | Age
                  | E Tag
                  | Location
                  | Proxy-Authenticate
                  | Retry-After
                  | Server
                  | Vary
                  | WWW-Authenticate
```

The name of the Response-header field can be extended reliably only in combination with a change in the version of the protocol.

Message Body

The response's message body may be referred to for convenience as a response body.

The body of the message is used for most responses. The exceptions are where a server is using certain status codes and where the server is responding to a client request, which asks for the headers but not the response body.

For a response to a successful request, the body of the message contains either some information about the status of the action which is requested by the client or the resource which is requested by the client. For the response to an unsuccessful request, the body of the message might provide further information about some action the client needs to take to complete the request successfully or about the reason for the error.

WORKING WITH COOKIES IN JAVA

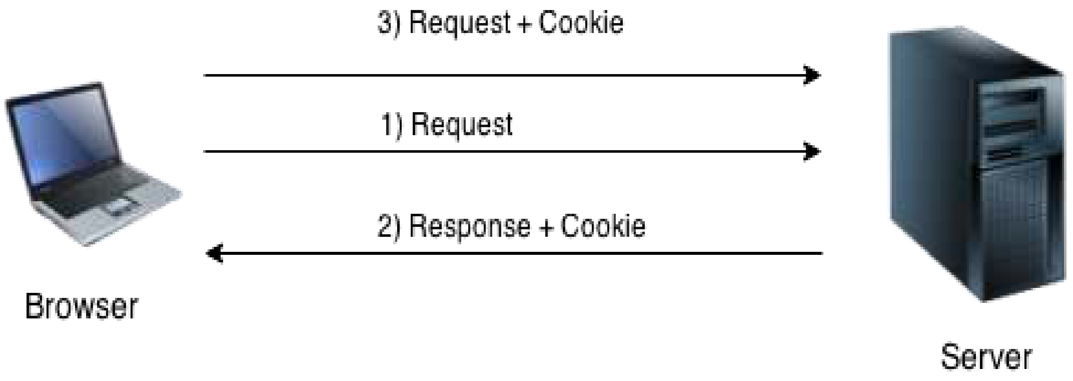
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session**. It is not removed each time when user closes the browser. It is removed only if user logout or sign out.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

Create a Cookie

Let's see the simple code to create cookie.

```
1.      Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
2.      response.addCookie(ck);//adding cookie in the response
```

Delete a Cookie

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
1.      Cookie ck=new Cookie("user","");//deleting value of cookie
2.      ck.setMaxAge(0);//changing the maximum age to 0 seconds
3.      response.addCookie(ck);//adding cookie in the response
```

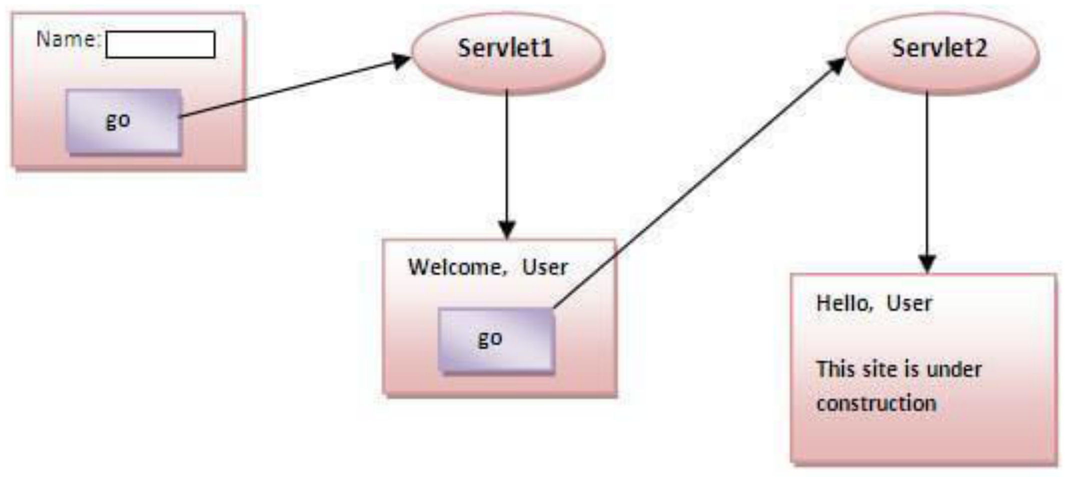
Get Cookies

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class FirstServlet extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response

    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");

    out.close();

        }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

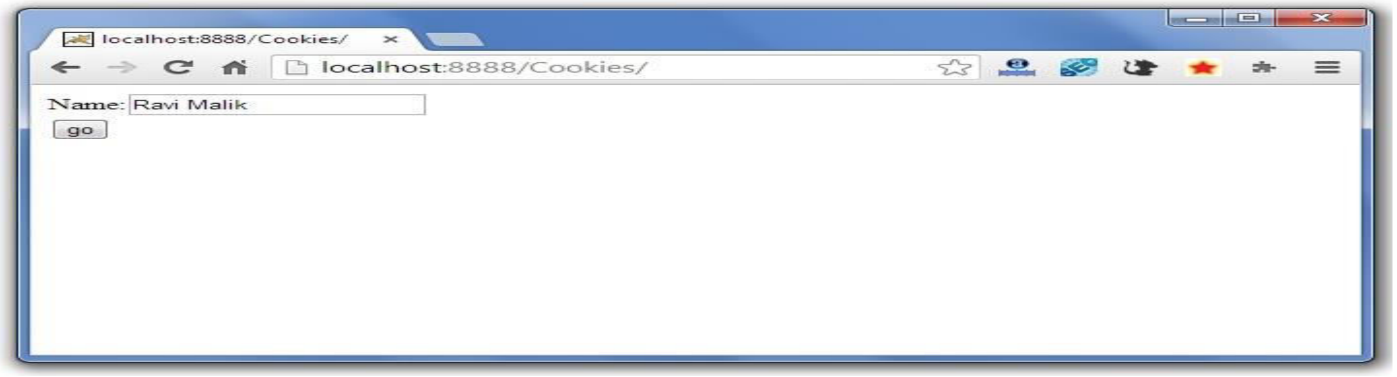
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());

    out.close();
```

```

        }catch(Exception e){System.out.println(e);}
    }


}
```

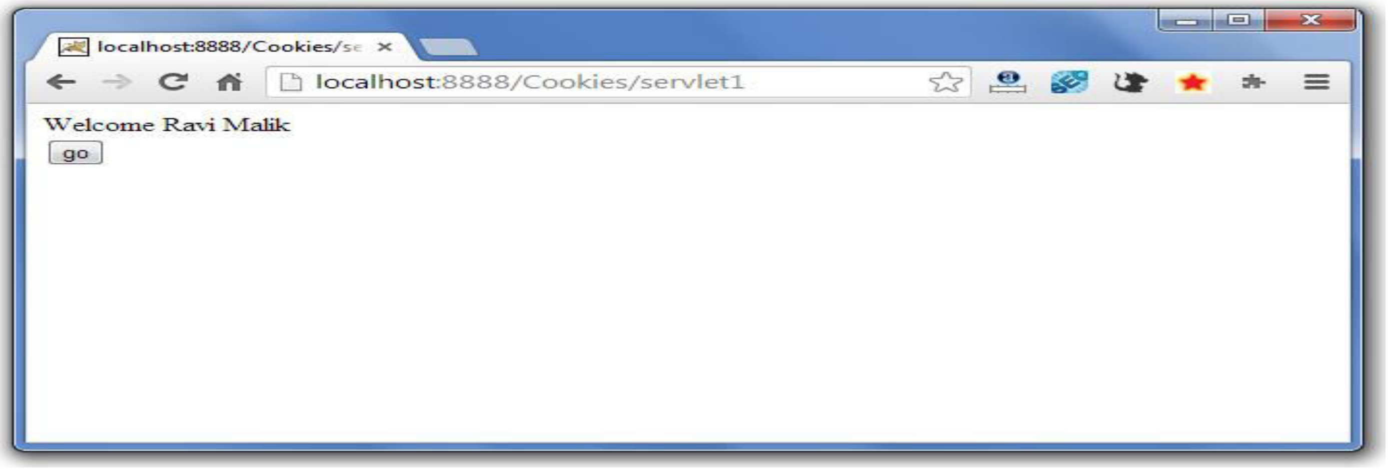
web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

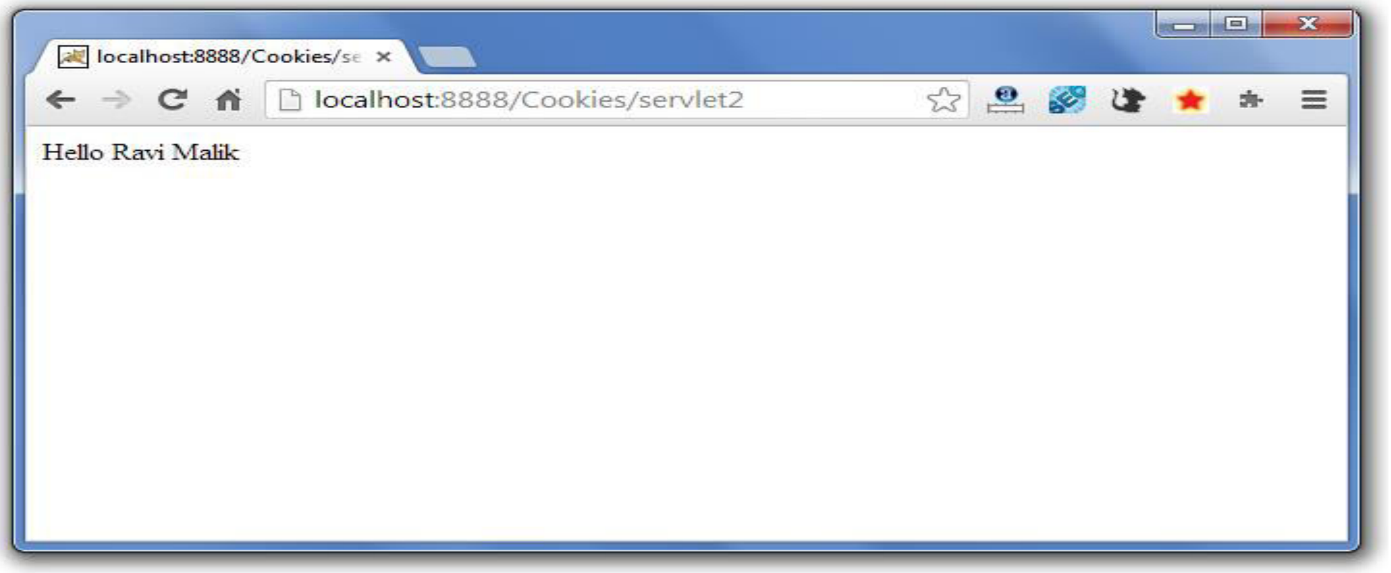
Output



Name: Ravi Malik



Welcome Ravi Malik
go



Hello Ravi Malik

JSP – <u>JAVA SERVER PAGES</u>

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, which makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

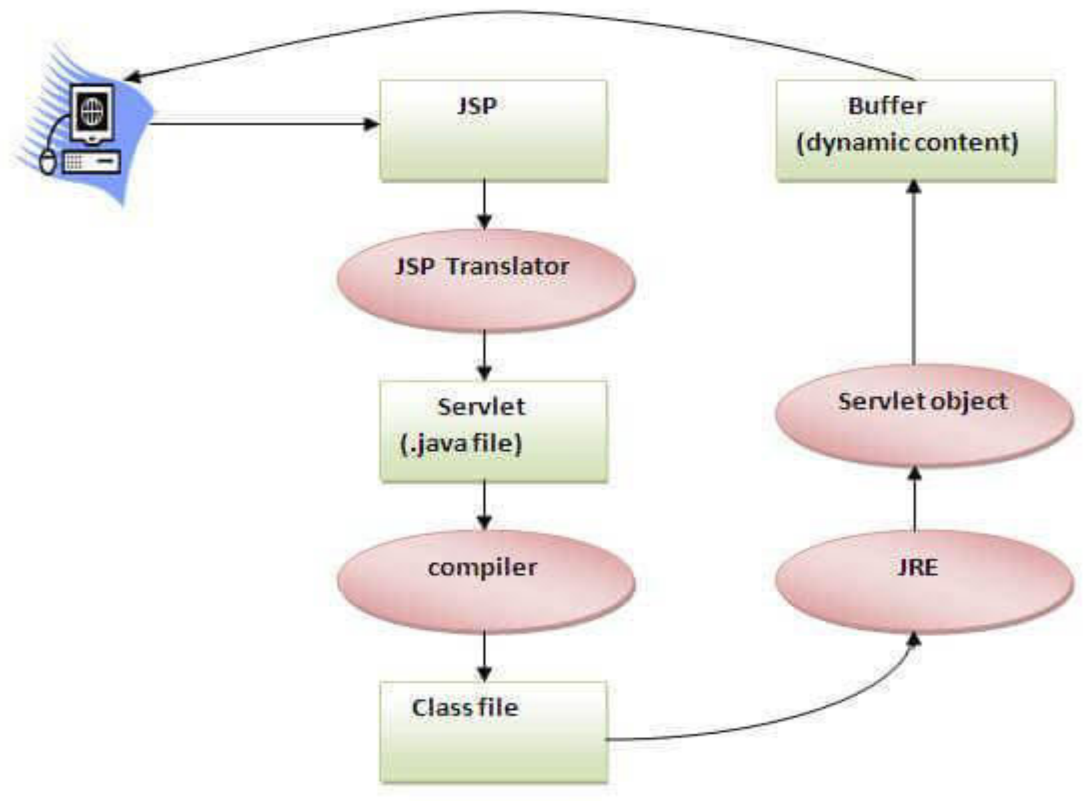
4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Class loading (the class loader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

Run a simple JSP Page

Follow the following steps to execute this JSP page:

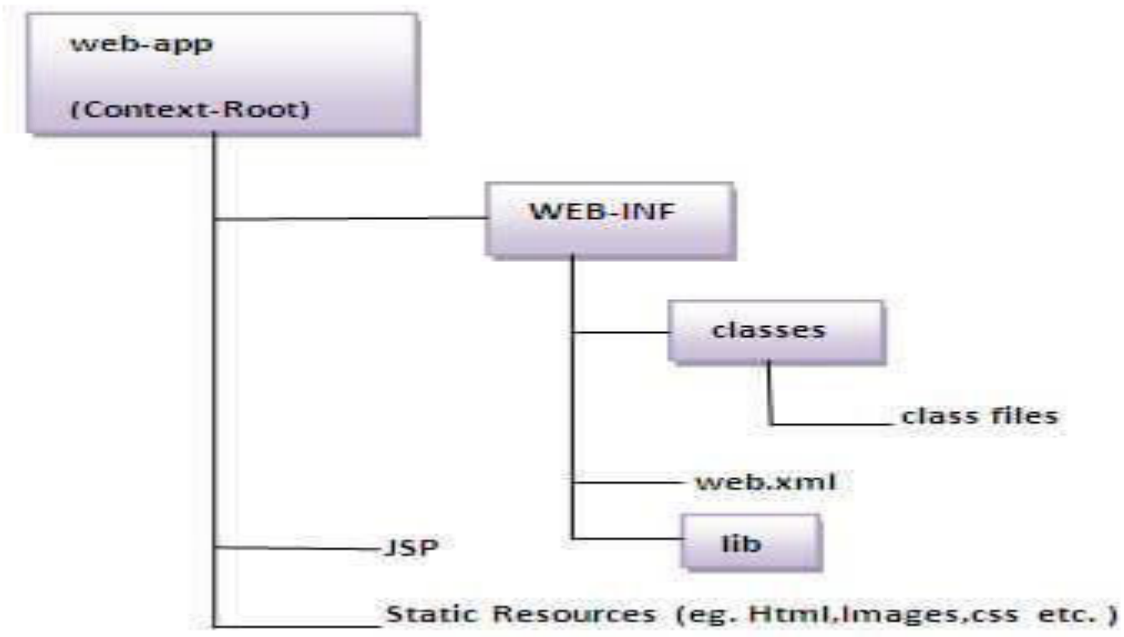
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Need to follow the directory structure to run a simple JSP

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



INSTALLATION

To install Java on your computer, follow these steps:

Visit the Java SE Downloads page (https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) and click on the "JDK Download" button.

Accept the license agreement and choose the appropriate version of Java Development Kit (JDK) for your operating system. If you're not sure which version to download, select the latest version.

Download the installer file for your operating system.

Once the download is complete, run the installer and follow the on-screen instructions to install Java.

After the installation is complete, open the command prompt (Windows) or terminal (Mac/Linux) and type "java -version" to verify that Java is installed correctly. You should see the installed version of Java printed in the console.

Congratulations! You have successfully installed Java on your computer.

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

1. <% java source code %>

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP Components

In this section we will discuss about the elements of JSP.

Structure of JSP Page:

JSPs are comprised of standard HTML tags and JSP tags. The structure of Java Server pages are simple and easily handled by the servlet engine. In addition to HTML , you can categorize JSPs as following -

- Directives
- Declarations
- Script lets
- Comments
- Expressions

Directives:

A directives tag always appears at the top of your JSP file. It is global definition sent to the JSP engine. Directives contain special processing instructions for the web container. You can import packages, define error handling pages or the session information of the JSP page. Directives are defined by using <%@ and %> tags.

Syntax -

<%@ directive attribute="value" %>

Declarations:

This tag is used for defining the functions and variables to be used in the JSP. This element of JSPs contains the java variables and methods which you can call in expression block of JSP page. Declarations are defined by using <%! and %> tags. Whatever you declare within these tags will be visible to the rest of the page.

Syntax -

<%! Declaration %>

Scriptlets:

In this tag we can insert any amount of valid java code and these codes are placed in _jsp Service method by the JSP engine. Scriptlets can be used anywhere in the page. Scriptlets are defined by using <% and %> tags.

Syntax -

<% Scriptlets%>

Comments:

Comments help in understanding what is actually code doing. JSPs provides two types of comments for putting comment in your page. First type of comment is for output comment which is appeared in the output stream on the browser. It is written by using the <!-- and --> tags.

Syntax -

```
<!-- comment text -->
```

Second type of comment is not delivered to the browser. It is written by using the <%-- and --%> tags.

Syntax -

```
<%-- comment text --%>
```

Expressions:

Expressions in JSPs is used to output any data on the generated page. These data are automatically converted to string and printed on the output stream. It is an instruction to the web container for executing the code with in the expression and replace it with the resultant output content. For writing expression in JSP, you can use <%= and %> tags.

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>

```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<%  java source code %>
```

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
```

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP DIRECTIVES

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- tag lib directive

Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

```
<%@ page attribute="value" %>
```

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding

- errorPage
- isErrorPage

1) Import

The import attribute is used to import class,interface or all the members of a package. It is similar to import keyword in java class or interface.

Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

2) Content Type

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
<html>
<body>

<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

---

3) Extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4) Info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

---

5) Buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page. The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

6) Language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7) IsELIgnored

We can ignore the Expression Language (EL) in jsp by the is ELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8) IsThreadSafe

Servlet and JSP both are multi-threaded. If you want to control this behavior of JSP page, you can use isThreadSafe attribute of page directive. The value of isThreadSafe value is true. If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

```
<%@ page isThreadSafe="false" %>
```

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9) Error Page

The error Page attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of error Page attribute

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

---

10) IsErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

JSP Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	**Jsp Declaration Tag**
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

A COMPLETE EXAMPLE FOR JSP

UNIT-V
ADVANCED TECHNIQUES

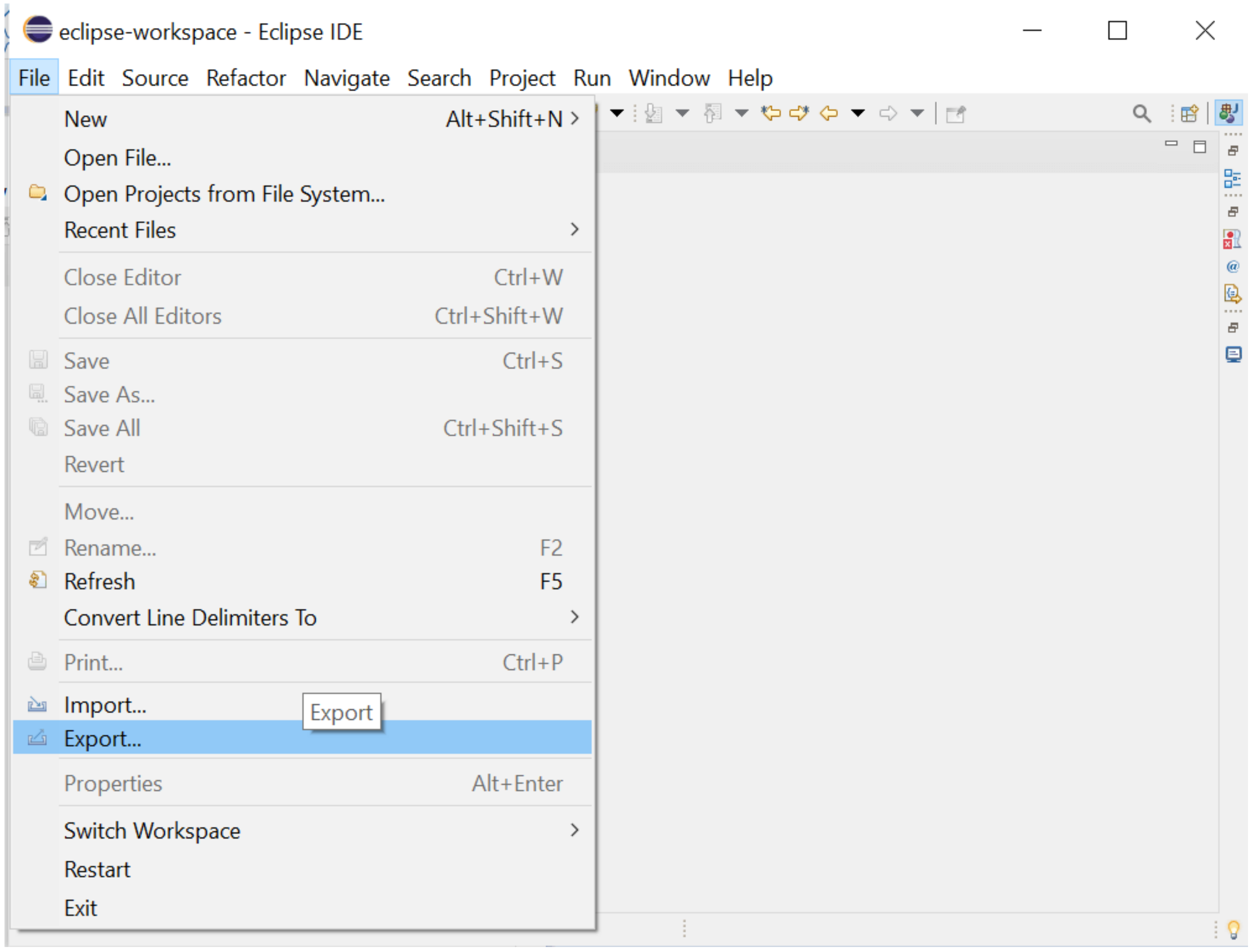
JAVA CREATE JAR FILES

In Java, JAR stands for Java ARchive, whose format is based on the zip format. The JAR files format is mainly used to aggregate a collection of files into a single one. It is a single cross-platform archive format that handles images, audio, and class files. With the existing applet code, it is backward-compatible. In Java, Jar files are completely written in the Java programming language.

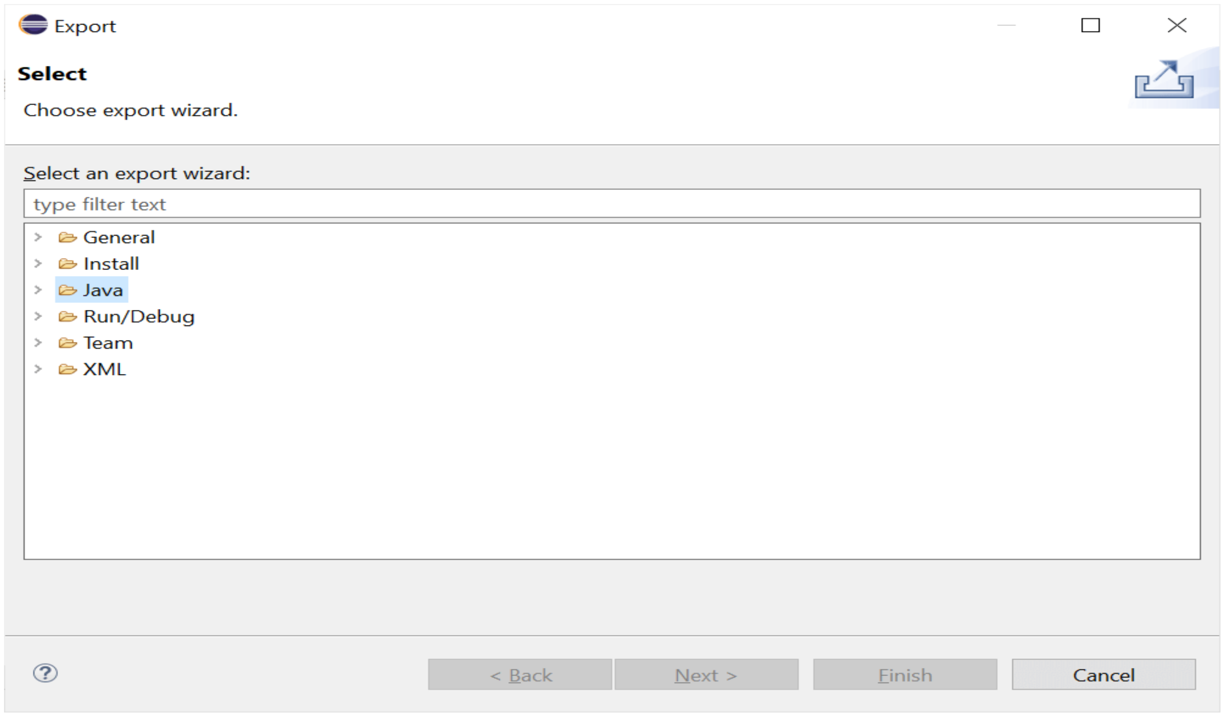
We can either download the JAR files from the browser or can write our own JAR files using **Eclipse** IDE.

The steps to bundle the source code, i.e., .java files, into a JAR are given below. In this section, we only understand how we can create JAR files using eclipse IDE. In the following steps, we don't cover how we can create an executable JAR in Java.

1. In the first step, we will open Eclipse IDE and select the **Export** option from the **File** When we select the Export option, the Jar File wizard opens with the following screen:



2.


Export

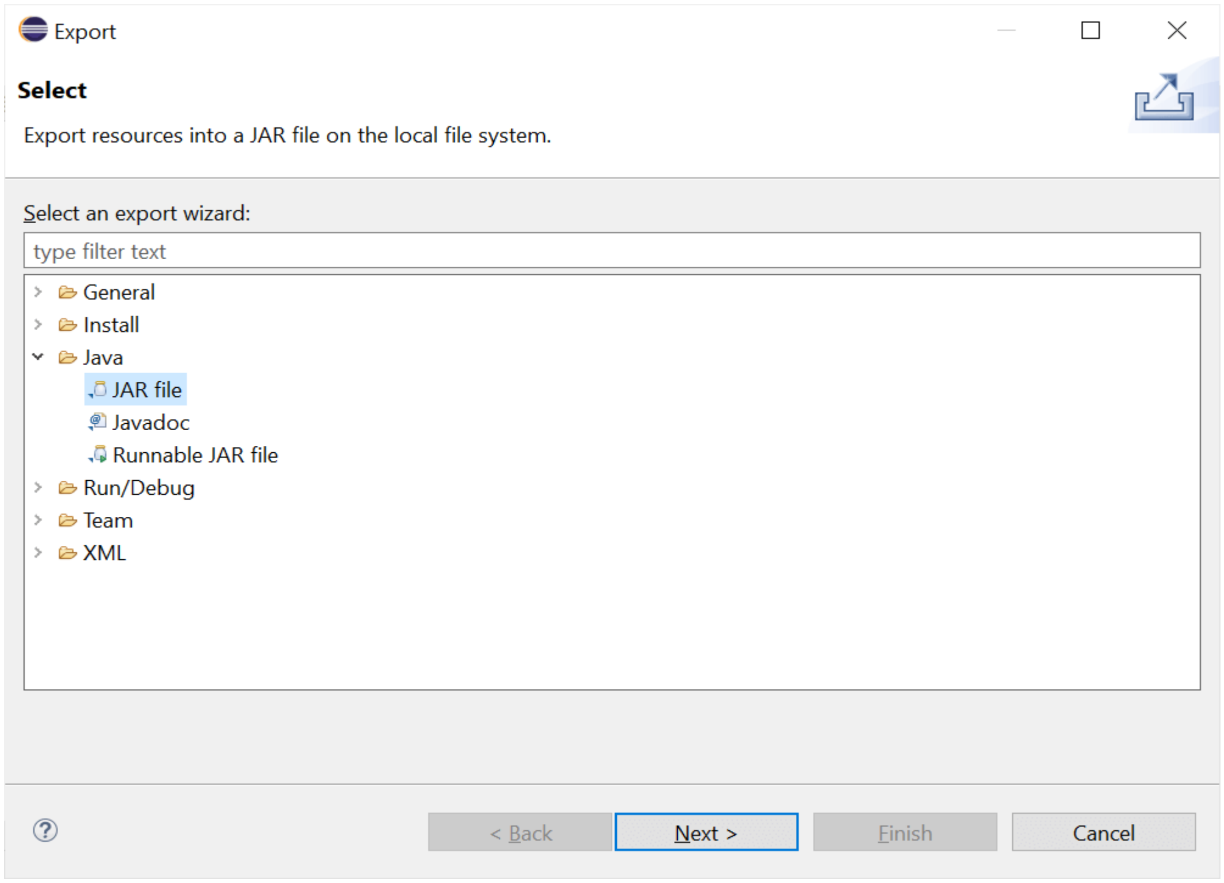
Select

Choose export wizard.

Select an export wizard:

type filter text

3. From the open wizard, we select the Java **JAR file** and click on the **Next** The Next button opens JAR Export for JAR File Specification.


Export

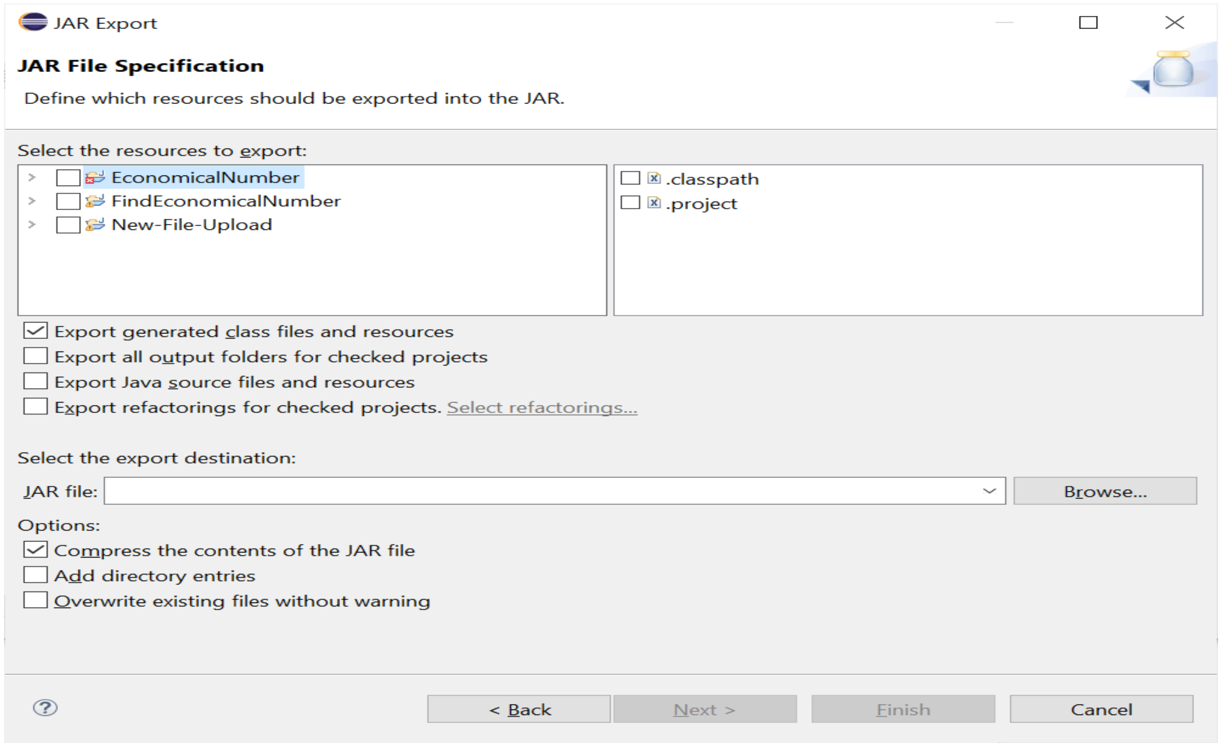
Select

Export resources into a JAR file on the local file system.

Select an export wizard:

type filter text

4.


JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [ ] EconomicalNumber
- [ ] FindEconomicalNumber
- [ ] New-File-Upload

- [ ] .classpath
- [ ] .project

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

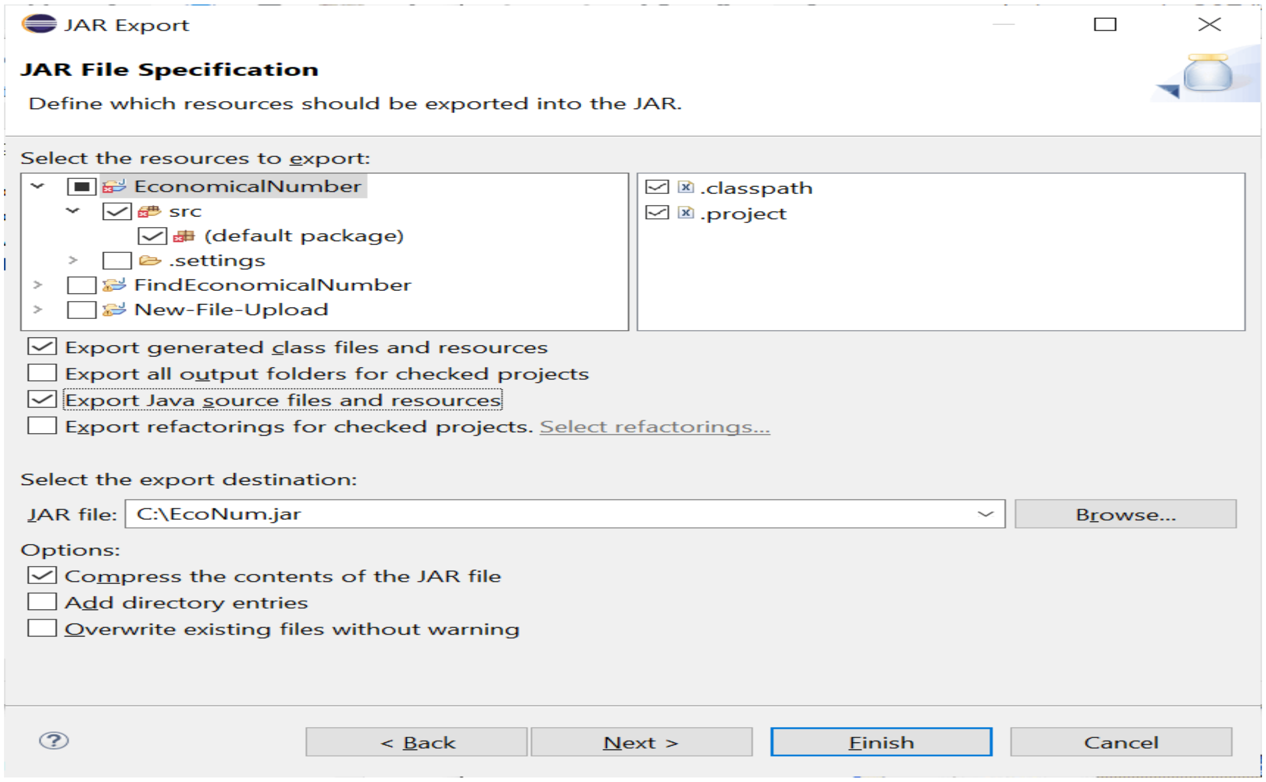
JAR file:

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

5. Now, from the JAR File Specification page, we select the resources needed for exporting in the **Select the resources to export** After that, we enter the JAR file name and folder. By default, the **Export generated class files and resources** checkbox is checked. We also check the **Export Java source files and resources** checkbox to export the source code.

6.


JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [■] EconomicalNumber
  - [✓] src
    - [✓] (default package)
  - [ ] .settings
- [ ] FindEconomicalNumber
- [ ] New-File-Upload

- [✓] .classpath
- [✓] .project

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [✓] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

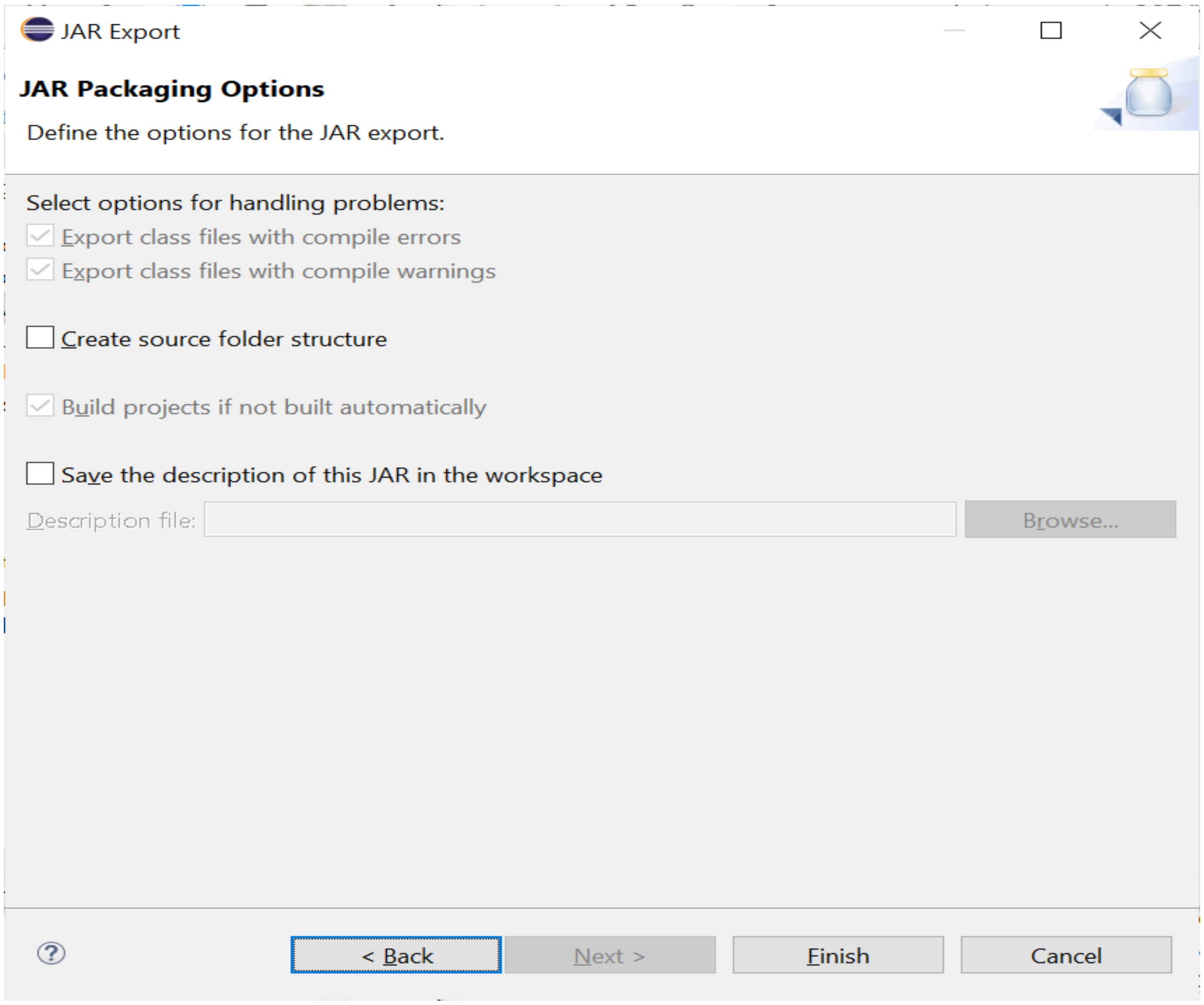
Select the export destination:

JAR file: C:\EcoNum.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

7. If there are other Java files or resources which we want to include and which are available in the open project, browse to their location and ensure the file or resource is checked in the window on the right.
8. On the same page, there are three more checkboxes, i.e., **compress the content of the JAR file, Add directory entries**, and **Overwrite existing files without warning**. By default, the **Compress content of the JAR file** checkbox is checked.
9. Now, we have two options for proceeding next, i.e., **Finish** and **Next**. If we click on the **Next**, it will immediately create a JAR file to that location which we defined in the **Select the export destination**. If we click on the **Next** button, it will open the Jar **Packaging Option** wizard for creating a JAR description, setting the advance option, or changing the default manifest.


JAR Export

JAR Packaging Options

Define the options for the JAR export.

Select options for handling problems:

- [✓] Export class files with compile errors
- [✓] Export class files with compile warnings
- [ ] Create source folder structure
- [✓] Build projects if not built automatically
- [ ] Save the description of this JAR in the workspace

Description file:

For now, we skip the **Next** and click on the **Finish** button.

10. Now, we go to the specified location, which we defined in the **Select the export destination**, to ensure that the JAR file is created successfully or not.

INTERNATIONALIZATION AND LOCALIZATION IN JAVA

Internationalization is also abbreviated as I18N because there are total 18 characters between the first letter 'I' and the last letter 'N'.

Internationalization is a mechanism to create such an application that can be adapted to different languages and regions.

Internationalization is one of the powerful concept of java if you are developing an application and want to display messages, currencies, date, time etc. according to the specific region or language.

Localization is also abbreviated as I10N because there are total 10 characters between the first letter 'L' and last letter 'N'. Localization is the mechanism to create such an application that can be adapted to a specific language and region by adding locale-specific text and component.

Understanding the culturally dependent data before starting internationalization

Before starting the internationalization, Let's first understand what are the information’s that differ from one region to another. There is the list of culturally dependent data:

- Messages
- Dates
- Times
- Numbers
- Currencies
- Measurements
- Phone Numbers
- Postal Addresses
- Labels on GUI components etc

Importance of Locale class in Internationalization

An object of Locale class represents a geographical or cultural region. This object can be used to get the locale specific information such as country name, language, variant etc.

Fields of Locale class

There are fields of Locale class:

1. public static final Locale ENGLISH
2. public static final Locale FRENCH

3. public static final Locale GERMAN
4. public static final Locale ITALIAN
5. public static final Locale JAPANESE
6. public static final Locale KOREAN
7. public static final Locale CHINESE
8. public static final Locale SIMPLIFIED_CHINESE
9. public static final Locale TRADITIONAL_CHINESE
10. public static final Locale FRANCE
11. public static final Locale GERMANY
12. public static final Locale ITALY
13. public static final Locale JAPAN
14. public static final Locale KOREA
15. public static final Locale CHINA
16. public static final Locale PRC
17. public static final Locale TAIWAN
18. public static final Locale UK
19. public static final Locale US
20. public static final Locale CANADA
21. public static final Locale CANADA_FRENCH
22. public static final Locale ROOT

Constructors of Locale class

There are three constructors of Locale class.

They are as follows:

1. Locale(String language)
2. Locale(String language, String country)
3. Locale(String language, String country, String variant)

Commonly used methods of Locale class

There are given commonly used methods of Locale class.

1. **public static Locale getDefault()** it returns the instance of current Locale
2. **public static Locale[] getAvailableLocales()** it returns an array of available locales.
3. **public String getDisplayCountry()** it returns the country name of this locale object.

4. **public String getDisplayLanguage()** it returns the language name of this locale object.
5. **public String getDisplayVariant()** it returns the variant code for this locale object.
6. **public String getISO3Country()** it returns the three letter abbreviation for the current locale's country.
7. **public String getISO3Language()** it returns the three letter abbreviation for the current locale's language.

Example of Local class that prints the information's of the default locale

In this example, we are displaying the information's of the default locale. If you want to get the information's about any specific locale, comment the first line statement and uncomment the second line statement in the main method.

```java
import java.util.*;
public class LocaleExample {
public static void main(String[] args) {
Locale locale=Locale.getDefault();
//Locale locale=new Locale("fr","fr");//for the specific locale

System.out.println(locale.getDisplayCountry());
System.out.println(locale.getDisplayLanguage());
System.out.println(locale.getDisplayName());
System.out.println(locale.getISO3Country());
System.out.println(locale.getISO3Language());
System.out.println(locale.getLanguage());
System.out.println(locale.getCountry());

}
}

```

```
Output:United States
       English
       English (United States)
       USA
       eng
       en
       US
```

Example of Local class that prints english in different languages

In this example, we are displaying english language in different language. Let's see how english is written in french and spanish languages.

```
import java.util.*;
public class LocaleExample2 {
   public static void main(String[] args) {
      Locale enLocale = new Locale("en", "US");
      Locale frLocale = new Locale("fr", "FR");
      Locale esLocale = new Locale("es", "ES");
      System.out.println("English language name (default): " +
                      enLocale.getDisplayLanguage());

      System.out.println("English language name in French: " +
                      enLocale.getDisplayLanguage(frLocale));
      System.out.println("English language name in spanish: " +
            enLocale.getDisplayLanguage(esLocale));
   }

}
```

Example of Local class that print display language of many locales

In this example, we are displaying the display lanuage of many locales.

```
import java.util.*;
public class LocaleEx {
public static void main(String[] args) {
Locale[] locales = { new Locale("en", "US"),
 new Locale("es", "ES"), new Locale("it", "IT") };

for (int i=0; i< locales.length; i++) {
 String displayLanguage = locales[i].getDisplayLanguage(locales[i]);
 System.out.println(locales[i].toString() + ": " + displayLanguage);
}
}

}
```

```
Output:en_US: English
       es_ES: espa?ol
       it_IT: italiano
```

Java Swing

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

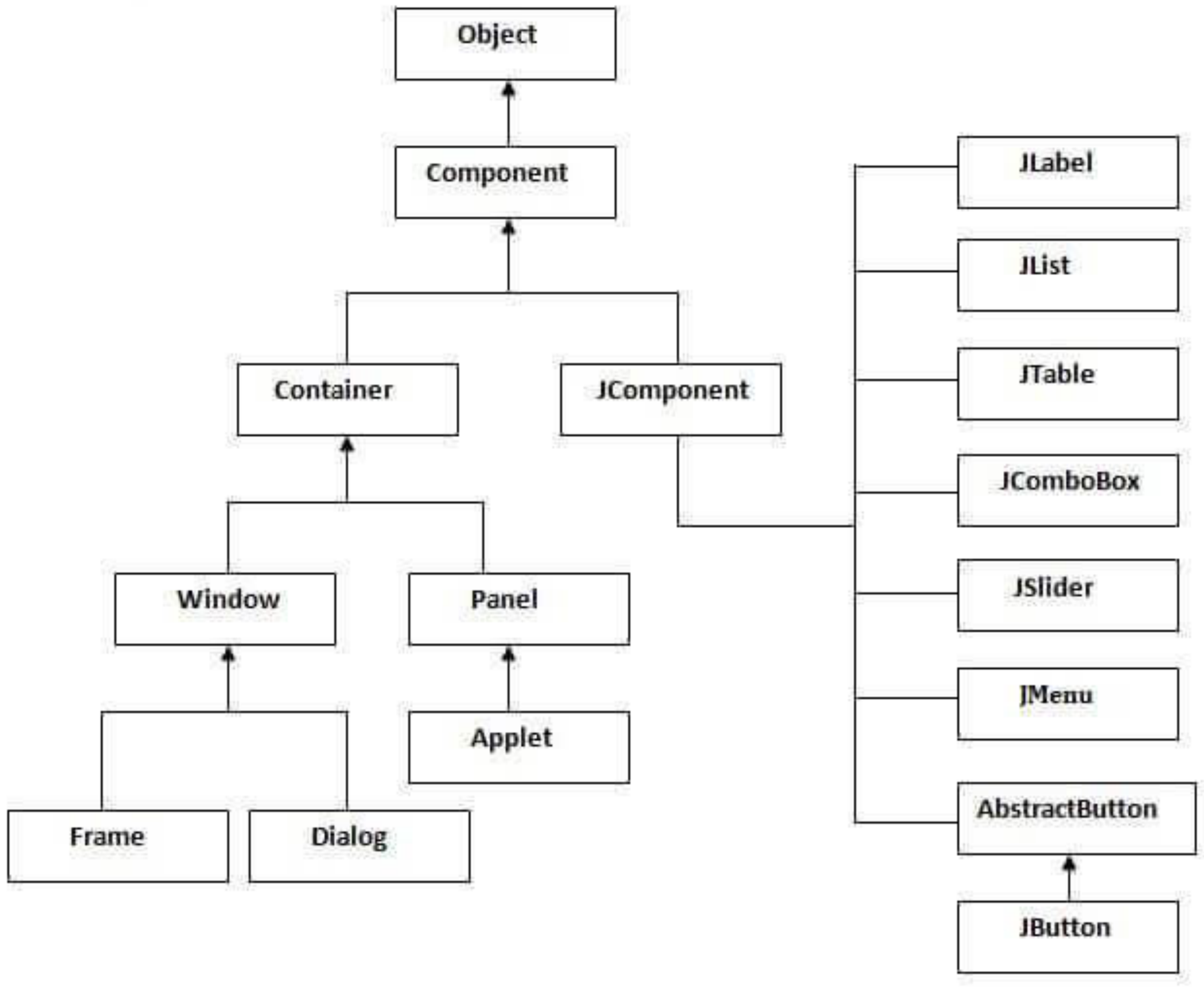
No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scroll panes, color chooser, tabbed pane etc.
5)	AWT **doesn't follows MVC** (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	Add a component on another component.
public void setSize(int width,int height)	Sets size of the component.
public void setLayout(LayoutManager m)	Sets the layout manager for the component.
public void setVisible(boolean b)	Sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

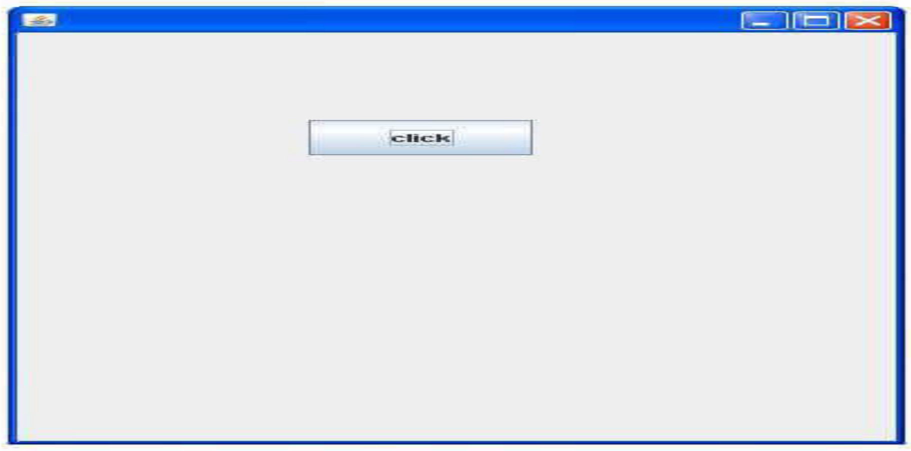
We can write the code of swing inside the main (), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main () method.

File: FirstSwingExample.java

```java
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
}
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;
public class Simple {
JFrame f;
Simple(){
f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}

public static void main(String[] args) {
new Simple();
}
}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

Advance Java

The dictionary meaning of **advance** is a forward movement or a development or improvement and the meaning of improve means thing that makes something better. All in all, we have to improve our basic knowledge to master in that particular field.

Java is divided into two parts i.e. **Core Java (J2SE)** and **Advanced Java (JEE)**. The core Java part covers the fundamentals (data types, functions, operators, loops, thread, exception handling, etc.) of the Java programming language. It is used to develop general purpose applications. Whereas **Advanced Java** covers the standard concepts such as database connectivity, networking, Servlet, web-services, etc. In this section, we will discuss **what is advance Java, its benefit, uses, topics of advance Java**, and the **difference between core Java and advance Java.**

It is a part of Java programming language. It is an advanced technology or advance version of Java specially designed to develop web-based, network-centric or enterprise applications. It includes the concepts like Servlet, JSP, JDBC, RMI, Socket programming, etc. It is a specialization in specific domain.

Most of the applications developed using advance Java uses tow-tier architecture i.e. Client and Server. All the applications that runs on Server can be considered as advance Java applications.

Why advance Java

- It simplifies the complexity of a building n-tier application.
- Standardizes and API between components and application sever container.
- JEE application Server and Containers provides the framework services.

Benefits of Advance Java

The four major benefits of advance Java that are, network centric, process simplification, and futuristic imaging standard.

- JEE (advance Java) provides libraries to understand the concept of **Client-Server architecture** for web- based applications.
- We can also work with web and application servers such as **Apache Tomcat** and **Glassfish** Using these servers, we can understand the working of HTTP protocol. It cannot be done in core Java.
- It is also important understand the advance Java if you are dealing with trading technologies like **Hadoop, cloud-native** and **data science**.
- It provides a set of services, **API** and **protocols**, that provides the functionality which is necessary for developing **multi-tiered** application, web-based application.
- There is a number of advance Java frameworks like, **Spring, Hibernate, Struts,** that enables us to develop secure **transaction-based** web applications such as banking application, inventory management application.

Difference between Core Java and Advance Java

Criteria	Core Java	Advance Java
Used for	It is used to develop general purpose application.	It is used to develop web-based applications.
Purpose	It does not deal with database, socket programming, etc.	It deals with socket programming, DOM, and networking applications.
Architecture	It is a single tier architecture.	It is a mute-tier architecture.
Edition	It is a Java Standard Edition.	It is a Java Enterprise Edition.
Package	It provides java.lang.* package.	It provides java.servlet.* package.

Advance Java Topics or Syllabus

1. Basics of a Web application

- HTTP protocol basics
- HTML language basics
- Need for a Web Container

2. Web Container and Web Application Project Set up

- To set up Tomcat Container on a machine
- To set up a Servlets JSP project in Eclipse
- To configure dependency of Servlet JSP APIs

- Web application project structure

3. Servlets

- HTTP Methods; GET, POST, PUT, DELETE, TRACE, OPTIONS
- GET/POST request; differences between the two
- Servlet Lifecycle
- Servlet Context and Servlet Configuration
- Forwarding and Redirection of requests

4. Session Management

- Session information passing between client and server
- Session information passing mechanisms - Cookies, Rewriting
- How to destroy a session

5. JSPs

- Introduction to JSP and need for JSPs
- Basic HTML tags
- JSP Lifecycle

6. JSP Elements

- Scriptlets
- Expressions
- Declarations
- Significance of above elements and fitment into the JSP Lifecycle
- Page Directive
- Include Directives
- Tag lib Directive

7. JSP Tag library

- JSP Standard Actions
- Expression Language
- JSTL basics and it's usage
- Need for Custom Tag Library
- Custom Tag Library implementation

Struts Framework (version 2.x)

1. Basics of MVC

- MVC Type1 and Type2 architecture
- Struts 1 overview
- Struts 1 and Struts 2 comparison

2. Struts 2 Architecture

- Architecture Diagram explanation of following components:
- Components of Model, Views and Controller in Struts Framework
- Interceptors
- Model/Action classes
- Value Stack
- OGNL
- Introduction to configurations; framework and application architecture
- Declarative and Annotations configuration approaches

3. Struts 2 set up and first Action class

- Download JAR files
- Struts 2 project build up and Configuration files
- To build Action class
- To intercept an HTTP request via Struts2 framework using Action class
- Defining data and business logic in Action class
- Preparing and Forwarding control to Views

4. Struts 2 Interceptors

- Responsibilities of an Interceptor
- Mechanism of Interceptor calling in Struts 2
- Defining Interceptors
- Defining Interceptor stacks
- Defining Custom Interceptors

5. Struts 2 Tag Library

- Introduction to tag library of Struts 2 and it's usage

6. Struts 2 Validations

- Validations using Validate able interface
- Workflow interceptor mechanism for validations
- Validations using Validate able interface
- Validation Framework introduction and architecture
- Validating user input with above two mechanisms

7. Struts 2 Tiles Frameworks

- Introduction to Tiles in a page
- Struts2 Tiles framework introduction
- Defining tiles.xml file
- Configuring pages for tiles
- A complete Tiles example with Struts2

Hibernate Framework (version 3.x)

- ORM implementations

2. Hibernate Architecture

- Introduction to Hibernate
- Hibernate Architecture

3. Hibernate CRUD

- Setting up Hibernate project
- Configuring all JARs and XML files
- Setting up connection to DB using Hibernate
- Performing basic CRUD operations using Hibernate API
- Object Identity; Generator type classes
- Using SQL with Hibernate
- Using HQL
- Using Criteria queries

4. Mapping Collections and Associations

- To define sets, mas, lists in Hibernate
- Association Mappings:
    1. One to one
    2. One to many
    3. Many to one
    4. Many to many
- Hibernate Caching
- Explanation of various caching mechanisms in Hibernate

5. Using Hibernate Annotations

- Sample example of using Hibernate Annotations

Spring Framework (version 3.x)

1. Introduction to spring

- Spring Architecture explanation and all its components

2. Introduction to all modules of Spring

- Spring Bean Factory
- Spring Application Context
- Spring DI
- Spring Integration; Spring messaging, Spring JMS
- Spring MVC
- Spring DAO

3. Setting up spring

- Setting up of Spring framework
- Download JARs
- Configure XML files

4. Dependency Injection

- Bean Wiring mechanisms in Spring

5. Spring AOP

- Implementation of Spring AOP

Spring Boot Framework (Version 2.x)

1. Introduction

- Spring Boot Introduction
- Spring Boot Version
- Spring vs Spring Boot vs Spring MVC
- Spring Boot Architecture

2. Creating Project

- Spring Initialize
- Download & Install STS IDE
- Spring Boot Example
- Spring Boot CLI
- Spring Boot Example-STS

3. Project Components

- Annotations
- Dependency Management
- Application Properties
- Starters
- Starter Parent
- Starter Web
- Starter Data JPA
- Starter Actuator
- Starter Test
- Devtools
- Multi Module Project
- Packaging
- Auto-Configuration

4. Tool Suite

- Hello World Example
- Project Deployment Using Tomcat

5. Spring Boot AOP

- AOP Before Advice
- AOP After Advice
- AOP Around Advice
- After Returning Advice
- After Throwing Advice

6. Spring Boot Database

- JPA
- JDBC
- H2 Database
- Crud Operations

7. Spring Boot View

- Thyme leaf View

8. Spring Boot Caching

- Cache Provider
- EhCaching

9. Spring Boot Misc.

- Run Spring Boot Application
- Changing Port
- Spring Boot Rest Example

Web Services: REST and SOAP

- Logging Framework: Splunk, Log4J, SLF4j
- Version-control system + repository hosting service: Git + Github